

AFRL-IF-RS-TR-1998-43

Final Technical Report

April 1998



GLOBAL RESOURCE MANAGEMENT

SRI International

**Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. D293**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19980608 037

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

DTIC QUALITY INSPECTED 3

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

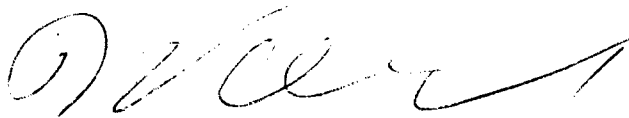
AFRL-IF-RS-TR-1998-43 has been reviewed and is approved for publication.

APPROVED:



THOMAS F. LAWRENCE
Project Engineer

FOR THE DIRECTOR:



WARREN H. DEBANY, JR.
Technical Advisor, Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

GLOBAL RESOURCE MANAGEMENT

Saurav Chatterjee, Bikash Sabata, Jaroslaw Sydir,
Ambatipudi Sastry, Wen Wang, Pamela Clark, Charles Hammond

Contractor: SRI International
Contract Number: F30602-95-C-0299
Effective Date of Contract: 1 November 1995
Contract Expiration Date: 28 February 1997
Program Code Number: 5E20
Short Title of Work: Global Resource Management
Period of Work Covered: Nov 95 - Feb 97

Principal Investigator: Ambatipudi Sastry
Phone: (415) 859-6141
AFRL Project Engineer: Thomas F. Lawrence
Phone: (315) 330-2925

Approved for public release; distribution unlimited.

This research was supported by the Defense Advanced Research
Projects Agency of the Department of Defense and was monitored by
Thomas F. Lawrence, AFRL/IFGA, 525 Brooks Road, Rome, NY
13441-4505.

| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 | |
|---|---|--|---|--|
| <small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small> | | | | |
| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE April 1998 | 3. REPORT TYPE AND DATES COVERED Final Nov 95 - Feb 97 | | |
| 4. TITLE AND SUBTITLE GLOBAL RESOURCE MANAGEMENT | | 5. FUNDING NUMBERS C - F30602-95-C-0299 PE - 62301E PR - D293 TA - 00 WU - 03 | | |
| 6. AUTHOR(S) Saurav Chatterjee, Bikash Sabata, Jaroslaw Sydir, Ambatipudi Sastry, Wen Wang, Pamela Clark, and Charles Hammond | | 8. PERFORMING ORGANIZATION REPORT NUMBER SRI Project 7173 ITAD-7173-FR-97-118 | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Ave Menlo Park CA 94025-3493 | | | | |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 N. Fairfax Drive Arlington VA 22203-1714 | | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-1998-43 | | |
| 11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Thomas F. Lawrence/IFGA/(315) 330-2925 | | | | |
| 12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited. | | | 12b. DISTRIBUTION CODE | |
| 13. ABSTRACT (Maximum 200 words) An increasing number of applications (e.g., collaborative planning, video conferencing, telemedicine, and distributed command and control) have quality-of-service (QoS) requirements including data precision, accuracy, timeliness, security and availability. These QoS requirements are generally end-to-end requirements because the users of the applications are interested in the end results. This report describes the architecture developed for the Global Resource Management (GRM) project, to enable adaptive, end-to-end, scalable resource management of distributed systems. The architecture includes a QoS taxonomy; models to capture the application, resource, and system design space; algorithms to navigate this system design space; and algorithms to evaluate points within this design space so as to determine the impact on application QoS and system metrics. The architecture can be realized in many system domains, including command and control, multimedia, and medical domains, to provide domain-specific application QoS support. | | | | |
| 14. SUBJECT TERMS Quality of Service, Distributed Systems, Adaptive Distributed Systems, Resource Management | | | 15. NUMBER OF PAGES 146 | |
| | | | 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 | |

CONTENTS

| | |
|--|----|
| ACKNOWLEDGMENTS | V |
| EXECUTIVE SUMMARY | VI |
| 1 INTRODUCTION | 1 |
| 1.1 PROJECT SUMMARY..... | 1 |
| 1.2 KEY PERSONNEL | 1 |
| 1.3 WHAT IS QOS-DRIVEN RESOURCE MANAGEMENT AND WHY IS IT IMPORTANT? | 2 |
| 1.4 RELATED WORK | 4 |
| 1.5 GRM PROJECT GOALS | 5 |
| 1.6 POTENTIAL APPLICATIONS OF QOS-DRIVEN RESOURCE MANAGEMENT | 5 |
| 1.7 ADAPTIVE QOS-DRIVEN RESOURCE MANAGEMENT ARCHITECTURE | 6 |
| 1.8 RESOURCE MANAGEMENT WITHIN A DISTRIBUTED SYSTEM ARCHITECTURE..... | 10 |
| 2 TAXONOMY FOR QOS SPECIFICATIONS | 13 |
| 2.1 INTRODUCTION | 13 |
| 2.2 BACKGROUND | 13 |
| 2.3 QOS TAXONOMY | 14 |
| 2.4 PERFORMANCE METRICS..... | 15 |
| 2.5 SECURITY..... | 18 |
| 2.6 RELATIVE IMPORTANCE | 18 |
| 2.7 POLICIES | 18 |
| 2.8 CONCLUSIONS | 19 |
| 3 APPLICATION, RESOURCE, AND SYSTEM MODELS | 21 |
| 3.1 RELATED WORK | 22 |
| 3.2 APPLICATION MODELS..... | 23 |
| 3.3 RESOURCE MODEL..... | 32 |
| 3.4 SYSTEM MODEL | 35 |
| 3.5 CORRESPONDENCE BETWEEN APPLICATION AND SYSTEM MODELS..... | 39 |
| 3.6 DISCUSSION | 41 |
| 4 ADAPTIVE RESOURCE MANAGEMENT ALGORITHMS | 43 |
| 4.1 QOS TRANSLATION..... | 45 |
| 4.2 INTEGRATED RESOURCE ALLOCATION AND ROUTING | 46 |
| 4.3 END-TO-END SCHEDULING..... | 47 |
| 4.4 END-TO-END QOS ANALYSIS..... | 48 |

| | | |
|----------|--|-----------|
| 4.5 | QOS-DRIVEN TRADE-OFFS | 48 |
| 4.6 | EXAMPLE | 49 |
| 4.7 | EXAMPLE TRANSLATION AND MAPPING | 52 |
| 5 | CORBA-BASED RESOURCE MANAGEMENT STRUCTURE | 59 |
| 5.1 | OVERVIEW OF CORBA..... | 59 |
| 5.2 | FEASIBILITY OF A CORBA-BASED RESOURCE MANAGEMENT STRUCTURE..... | 60 |
| 5.3 | APPLICATION OF CORBA CONCEPTS TO GRM MODELS..... | 62 |
| 5.4 | CHALLENGES..... | 62 |
| 5.5 | RELATED WORK..... | 63 |
| 5.6 | AREAS OF FUTURE WORK..... | 65 |
| 6 | RESOURCE MANAGEMENT FOR TELESEMINAR..... | 67 |
| 6.1 | USER QOS REQUIREMENTS..... | 67 |
| 6.2 | APPLICATION MODEL | 68 |
| 6.3 | SYSTEM MODEL | 70 |
| 6.4 | QOS MAPPING | 71 |
| 6.5 | CONCLUSION..... | 79 |
| 7 | CONCLUSION..... | 81 |
| 7.1 | ACCOMPLISHMENTS | 81 |
| 7.2 | PUBLICATIONS AND PRESENTATIONS | 83 |
| 7.3 | FUTURE WORK..... | 84 |
| | REFERENCES | 85 |

Appendix A
OOA RESOURCE MODEL OBJECT SPECIFICATION

Appendix B
RESOURCE MODEL RELATIONSHIP SPECIFICATION

Appendix C
RATE-CONTROLLED STATIC-PRIORITY QUEUING DISCIPLINE

FIGURES

| | | |
|----|---|----|
| 1 | Resource Management Architecture | 9 |
| 2 | Adaptive Resource Management | 11 |
| 3 | A Taxonomy for QoS Specifications..... | 14 |
| 4 | QoS Architecture for Resource Management | 21 |
| 5 | Logical Application Stream Model | 26 |
| 6 | Recursive LASM..... | 28 |
| 7 | System-Specific Application Model Example | 30 |
| 8 | Video-Only QoS Benefit Function | 31 |
| 9 | Example System with Its Component Subsystems | 37 |
| 10 | Hierarchical Graph Structure Representing the Example System, Derived from Resource and System Topology..... | 37 |
| 11 | Subsystem or Resource Interfaces | 38 |
| 12 | Mapping of An Application (LROs) to Resources and Subsystems | 40 |
| 13 | Resource Management Architecture..... | 43 |
| 14 | Example Logical Application Stream Model..... | 50 |
| 15 | Example System | 51 |
| 16 | Composed Application | 53 |
| 17 | Vertical and Horizontal Translation and Mapping Perspectives | 54 |
| 18 | CORBA Organization and Components..... | 59 |
| 19 | CORBA Client-Server Request | 60 |
| 20 | Decomposition of the Teleseminar Service..... | 69 |
| 21 | Decomposition of the Video Service | 69 |
| 22 | Decomposition of the Video I/O Service..... | 70 |
| 23 | Video Service Decomposition | 70 |
| 24 | Example System for Teleseminar Application..... | 71 |
| 25 | Delay Aggregation Example of QoS Translation..... | 72 |
| 26 | Network Configuration Used for QoS Mapping | 73 |

TABLES

| | | |
|----|---|----|
| 1 | LUoW Attributes | 27 |
| 2 | PUoW Sets..... | 30 |
| 3 | Example PUoWs | 51 |
| 4 | Representative QoS Requirements..... | 55 |
| 5 | Network-Layer QoS Parameters in Relation to Application QoS Requirements | 57 |
| 6 | Translation of User QoS to Application Video QoS..... | 67 |
| 7 | Translation of User QoS to Application Audio QoS..... | 68 |
| 8 | Translation of User QoS to Application Whiteboard QoS..... | 68 |
| 9 | QoS Translation from Network Layer to Application Layer | 74 |
| 10 | QoS Translation from Application Layer to Network Layer | 74 |
| 11 | Translation of Network-Layer QoS to AAL5-SAR QoS and to ATM QoS..... | 78 |

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of the following individuals, without which this report could not have been written:

- Ms. Elin Klaseen, SRI International
- Dr. Michael Davis, Perspecta Inc.

EXECUTIVE SUMMARY

An increasing number of applications (e.g., collaborative planning, video conferencing, telemedicine, and distributed command and control) have quality-of-service (QoS) requirements including data precision, accuracy, timeliness, security and availability. These QoS requirements are generally end-to-end requirements because the users of the applications are interested in the end results. For example, the user of a video-on-demand application is interested in the quality of the picture (the end result), which is an aggregation of the QoS of retrieving the video from the database, transmitting it over the network, decompressing it, and displaying it.

Developing a distributed system that can provide this type of end-to-end QoS support to each application is difficult, because the different resources used by the applications must be managed in a coordinated manner. This difficulty is exacerbated by the heterogeneous nature of today's systems (i.e., these systems comprise heterogeneous computing, storage, and communication resources), the high levels of resource sharing among applications with diverse QoS requirements, and the constant change in the system state (i.e., continuous arrivals and departures of applications, resource failures and recoveries, and application QoS change requests). These characteristics make systems highly unpredictable; QoS-driven resource management is required to make the system more predictable, to manage resources efficiently, and to provide QoS support for each application.

Continuous system state changes also motivate the need for *adaptive* QoS-driven resource management. Adaptivity denotes that the resource manager dynamically *reallocates*, *reschedules*, and *restructures* applications so as to maximize the QoS that can be provided to applications in the new system state. If the new system state restrains the system from maintaining each application's QoS level, the resource manager must then *gracefully degrade* the affected applications' QoS in a manner that minimizes the adverse impact on each of the application users.

This report describes the architecture that SRI International and Stanford Telecommunications, Inc. developed for the Global Resource Management (GRM) project,^{*} to enable adaptive, end-to-end, scalable resource management of distributed systems. The architecture includes a QoS taxonomy; models to capture the application, resource, and system design space; algorithms to navigate this system design space; and algorithms to evaluate points within this design space so as to determine the impact on application QoS and system metrics. The architecture can be realized in many system domains, including command and control, multimedia, and medical domains, to provide domain-specific application QoS support.

Our QoS taxonomy formally defines the dimensions of QoS and the relationship between these QoS metrics. Although people think of the various components of QoS (such as functional requirements, real time requirements, fault tolerant requirements, and security requirements) as separate, we show that these requirements can be elegantly captured by means of a small number of QoS parameters. Our QoS taxonomy also defines the relationship between the different QoS metrics, enabling a greater understanding of how changes in one QoS metric affect other QoS metrics.

^{*}This project was conducted under Contract F30602-95-C-0299 with Rome Laboratory, during an 18-month period beginning 1 November 1995.

Resource management involves managing three somewhat distinct “parties” with potentially conflicting objectives. We say that each of these parties views resource management from a different perspective. One of the unique components of our work is that we consider all three of these perspectives when performing resource management. These three perspectives, and the models used to capture each perspective, are central components of our work; we summarize each below.

Each application is concerned with the level of QoS it receives; but not with how this QoS level affects other applications. We call this point of view the *application perspective*; we model it by means of two abstractions. The first application model, denoted the *Logical Application Stream Model*, recursively captures a distributed application’s structure, resource requirements, and relevant end-to-end QoS parameters. The model enables the resource manager to structure the end-to-end application, allocate resources to the application, and to schedule the application on these resources. Later, when system state changes, the model enables the resource manager to restructure, reallocate, and reschedule the application dynamically. This recursive model enables application developers to elegantly model large-scale applications. The recursive model also enables the application to be ported to many different systems.

The second application model, denoted the *Benefit Function*, communicates the application user’s QoS preferences to the resource manager. When the total application load exceeds the system resource capabilities, the resource manager can use this information to gracefully degrade application QoS such that the adverse effect on each application user is minimized.

The individual system resources (such as processors or communication networks) also have a perspective on resource management. We call this the *resource perspective*. This perspective is local to the management of each resource because each resource is unconcerned about other resources or applications running on other resources. Our Resource Model captures a resource’s attributes and scheduling behavior within a “black box” model, exporting only a uniform QoS interface. This approach enables us to model heterogeneous resources uniformly; it also enables the resource management scheme to provide end-to-end QoS support to applications running on these heterogeneous resources.

The system is composed of resources that support a set of applications. Providing end-to-end QoS support to these applications, while utilizing these resources efficiently, is the responsibility of the system and the system-wide resource manager. Because applications and resources themselves have conflicting goals, the system also includes system policies and algorithms to resolve these conflicts. Examples of such policies are end-to-end scheduling policies; policies to decide which application’s QoS is to be degraded when there are not enough resources to provide the desired QoS to all applications; admissions control policies; and policies that govern the amount of effort and time that should be expended in attempting to find the optimal resource allocation. Algorithms within the system-wide resource manager include the following: QoS translation, analysis, scheduling, allocation, and trade-offs. We call this the *system perspective*.

Our hierarchical System Model captures the set of resources comprised by the system, the topological and administrative structure of the system, and the policies and objectives that govern each of these administrative domains. An administrative domain comprising a set of resources is

denoted a subsystem. The hierarchical model enables us to model large systems of systems comprising many local resource management schemes. The hierarchical model also prunes the search space when the resource manager allocates resources to applications.

In the report, we also propose a road map for the development of the resource management algorithms, describe previous work in each area, and describe the missing elements needed for end-to-end resource management, in the contexts of QoS translation, allocation and routing, scheduling, analysis and trade-offs.

1 INTRODUCTION

1.1 PROJECT SUMMARY

SRI International (SRI) and Stanford Telecommunications, Inc. (STel),* under contract with Rome Laboratory (RL), have developed a conceptual framework for adaptive, integrated enterprise management. Such an enterprise management system will control and allocate the resources of the next generation of command, control, communications, computers, and intelligence (C⁴I) enterprises, in accordance with the users' quality-of-service (QoS) requirements, and will consider changes in operational modes, variations in the availability of resources, mobility, and anomalies. Such an adaptive, global resource management system will enable large, heterogeneous, distributed C⁴I systems to (1) transition from peacetime operation to crisis response without any interruption of service; (2) utilize limited system resources effectively and efficiently; and (3) continue to operate with some level of functionality despite failures and anomalies.

This effort, known as the Global Resource Management (GRM) project, began on 1 November 1995, and has an 18-month period of performance, under Contract F30602-95-C-0299. The GRM study is an extension of the System Resource Management (SRM) project, also sponsored by RL.

This final technical report summarizes the work performed during the entire project, in fulfillment of contract data requirements list (CDRL) item A005. In the remainder of Section 1 we describe the motivation for our work, describe related work, and present an overview of our technical approach. In Section 2 we describe our QoS Taxonomy, and in Section 3 we describe the application, resource, and system models that are the cornerstones of our architecture. In Section 4 we address the translation of QoS parameters and present an example from the networking domain. In Section 5 we describe how CORBA[†] may be used for resource management. In Section 6 we describe how our resource management (algorithms, models, and taxonomy) can be used together for a teleseminar application. In Section 7, we conclude by summarizing our accomplishments and discussing future work. Appendices A and B specify the objects and relationships, respectively, that appear in the Resource Model.[‡] Appendix C discusses the use of buffers to absorb and eliminate jitter at the network layer.

1.2 KEY PERSONNEL

The key SRI personnel for the GRM project are Dr. Michael S. Frankel, (the SRI project supervisor); Dr. Ambatipudi Sastry (the SRI project leader); Dr. Saurav Chatterjee; Dr. Bikash Sabata; and Mr. Jerry Sydir. The key personnel from STel are Mr. Charles Hammond (the STel project leader); Ms. Hien Nguyen; Ms. Pamela Clark; and Ms. Wen Wang. Mr. Thomas Lawrence of RL is the government project leader.

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

[†]CORBA: Common object request broker architecture.

[‡]Subsection 3.3 introduces the Resource Model.

1.3 WHAT IS QOS-DRIVEN RESOURCE MANAGEMENT AND WHY IS IT IMPORTANT?

Reductions in cost and increases in the availability of high-performance processors, high-capacity storage devices, and high-speed networks have made the distributed system computing infrastructures an economically and technically viable alternative to centralized systems. Distributed systems often comprise heterogeneous components and can span multiple administrative or management domains. Moreover, various local and distributed applications compete for system's resources. The environment encountered by these applications is very unpredictable because of random occurrences such as failures, security attacks, and applications' arrivals and departures.

All applications must meet *quality-of-service* requirements. While some applications are required only to provide functionally correct results (i.e., meet requirements for data *accuracy* and *precision*), an increasing number must also comply with *timing* (e.g., latency, jitter, or synchronization), *security*, and *availability* constraints. These QoS requirements are *end to end*, because the users are interested in end-to-end results. For example, the user of a video-on-demand application is interested in the quality of the picture (an end result), which is an aggregation of the QoS of retrieving the video from the database, transmitting it over the network, decompressing it, and displaying it.

Adaptive QoS-driven resource management is needed, to enable the applications to cope with a system's unpredictability in its attempts to meet their QoS requirements. This management effort must be a coordinated effort involving all system components at all levels, including that of the applications themselves. The end-to-end nature of QoS requirements necessitates the coordination of different system components (e.g., processing, storage, and communication components). The different trade-offs that can be made at the different system layers (e.g., resources, middleware, applications) must be coordinated in order to achieve a coherent resource management policy.

The concept of building applications that understand the QoS that they provide, and can adapt to varying levels of available system resources in attempting to provide a given level of QoS to their users, is not new. Video and audio applications that run on the Internet are designed to be adaptive to the Internet's unpredictable nature. However, these applications have no opportunity to coordinate the trade-offs that they make, with the distributed system on which they run (the Internet). We believe that coordinating the trade-offs that can be made within the application with the trade-offs that can be made within a distributed system will be a very powerful and important step towards achieving end-to-end QoS assurances within the system.

Consider as an example a next-generation navy ship in which a variety of computing and communications services are provided by a shared infrastructure. A land-based physician is performing telesurgery on a member of the ship's crew at sea. At the same time, the ship's officers are involved in an urgent video teleconference with the fleet commander. An attack renders enough of the ship-to-shore communications bandwidth unavailable that the needs of both the telesurgery and teleconference applications cannot be met simultaneously. Clearly, the system must reallocate the available resources. Under a conventional, priority-based resource allocation scheme, the less important application (in this case, telesurgery) would be dropped or would receive only best-effort service, a solution that is not ideal.

A better solution is to gracefully degrade the QoS of the two applications. The system may find, based on information conveyed from the user to the system, that the video teleconference application, although more important, is more flexible to QoS changes. One solution might be to perform better compression of the video signal, thus trading off computation for communications bandwidth. Another solution is to degrade the video from color to black and white, or to reduce the frame rate, in order to free up communications bandwidth for the telesurgery application. The resource manager can choose a combination of these and other solutions, based on the resource consumption of the various applications, their relative importance, and the sensitivity of the applications (and their users) to changes in QoS. Adaptivity results in a more efficient system that can support more applications and meet their QoS requirements.

This example illustrates the use of adaptive QoS-driven resource management to react to the loss of some available resources. The same types of trade-offs can be made when new application requests are submitted to the system or when the QoS requirements of an existing application change.

In order to be complete, we believe that a QoS-driven resource management system must possess the following characteristics:

- **Use a Comprehensive Definition of QoS.** Applications have multiple QoS requirements, i.e., timing, precision, accuracy, security, and availability [Sabata et al. 1997], that must be supported by the resource management scheme. By adopting a comprehensive definition of QoS, the resource manager can explicitly make resource management trade-offs between the various aspects of QoS such as performance, security, and availability. A comprehensive QoS definition includes the relationships between the trade-offs that can be made at various levels, allowing them to be made in a coordinated fashion.
- **Provide End-to-End QoS Support over Heterogeneous Resources.** Most distributed applications run over a number of computing, storage, and communication resources. To provide end-to-end QoS support to each application and to utilize resources efficiently, the resource management scheme must manage all of these resources uniformly. End-to-end QoS support would not be possible if only a subset of the resources supported QoS. For example, QoS support from the network, but not from the computing and storage resources, would be useless to the application, because a bottleneck at the computing or storage resources would deprive the application of its required QoS, no matter what the network does.
- **Provide Integrated QoS Support for Hard, Soft, and Non-Real Time Applications.** In order to amortize operational costs, a distributed system must multiplex the execution of many applications over shared system resources. These applications have diverse structures (continuous streams, e.g., audio, video; and client-server, e.g., remote banking), and have diverse QoS requirements (some applications require hard QoS assurances, e.g., telesurgery; others require soft QoS assurances, e.g., video on demand; and still others can tolerate best-effort service, e.g., electronic mail [e-mail]). The resource manager must provide integrated QoS support to all of these applications on a single system infrastructure, providing appropriate QoS assurances to real-time applications without unduly penalizing best-effort applications.

- **Be Adaptive to Changing System Conditions.** The state of a large distributed system is perpetually in flux, due to frequent arrivals and departures of applications, resource failures and recoveries, security attacks, and applications' QoS change requests. Static QoS support, where the allocation and scheduling decisions are made at connection time, leads to inefficient usage of resources and high operational costs. A better approach is to provide adaptive QoS support, where the resource manager changes allocation and scheduling decisions or gracefully degrades application QoS, in response to changes in system state.

1.4 RELATED WORK

Most previous work in QoS-driven resource management has focused on either the network or operating system layer. Resource management at each of these layers has been performed separately, with very little understanding of how the confluence of resource management at the different layers can provide end-to-end QoS support to applications. As a result, these disjoint approaches to resource management cannot provide such support. A significant amount of work concerning QoS has focused on network traffic: the Integrated Service Packet Network [Clark, Shenker and Zhang 1992]; the Flow Protocol [Zhang 1989]; the Tenet scheme [Ferrari and Verma 1990]; the HeiRAT scheme [Vogt, Herrtwich, and Nagarajan 1992]; the Session Reservation Protocol [Anderson, Herrtwich, and Schaefer 1990]; and transport-layer QoS considerations [Cambell et. al. 1993; Campbell, Coulson, and Hutchinson 1997]. The most comprehensive of these is the Tenet protocol [Ferrari and Verma 1990]. However, this work has been limited to temporal QoS issues. A good survey of the meaning of QoS at each layer is presented by Vogel et al. [1995]. Ferrari, Rameakeers, and Ventre [1992] discuss the interface between the communications system and the applications.

Work has been done on the allocation and scheduling of real-time tasks on a set of CPUs. Most of this work, however, has been confined to the single-resource domain [Jeffay 1993; Mercer, Savage, and Tokuda 1993] and is not applicable to QoS support for applications executing over multiple resources. Most of the published algorithms for end-to-end support involve hybrid off-line and/or on-line algorithms [Bettati and Liu 1990] that cannot be utilized for large-scale dynamic systems comprising soft, hard, and non-real time applications, where resources fail and recover, arrive and depart, and request changes in their QoS. Most of this work is also limited to the examination of temporal QoS issues such as latency and jitter. None of it considers uniform support for temporal QoS over computing, storage, and communication resources.

Recently, a few QoS architectures have been proposed in the literature. A good survey of the existing QoS architectures can be found in a review by Campbell, Aurrecoechea, and Hauw [1996]. Nahrstedt and Steinmetz [1993, 1995] specify the requirements for a QoS architecture for multimedia systems. Nahrstedt argues for an application-driven approach to QoS, but it is unclear how her approach could support application-level QoS adaptation and graceful degradation. Her approach also does not include integrated support for hard, soft, and non-real time applications. Zinky, Bakken, and Schantz [1997a] integrate CORBA with QoS attributes. It is, however, unclear whether their approach can be utilized to provide end-to-end QoS guarantees; nor does it support continuous-stream applications such as audio and video. QoS adaptivity, trade-off issues, and QoS-driven fault handling also are not reported by Nahrstedt and Smith [1993, 1995] or by Zinky, Bakken, and Schantz [1997a].

Campbell et al. [1993] and Campbell, Coulson, and Hutchinson [1997] were among the first to recognize the need for an integrated approach to resource management. They presented an integrated framework that deals with end-to-end application QoS requirements. The notion of flow is introduced as an important abstraction within the framework. Flow is defined to characterize the production, transmission, and consumption of the data associated with a single medium. Their QoS-A architecture includes the application layer, but does not specify any abstractions for understanding the QoS trade-offs that must be made within the application.

1.5 GRM PROJECT GOALS

Most previous work in the area of QoS addresses aspects of QoS in isolation. Even the attempts at defining QoS architectures do not address all of the requirements that we outlined in Subsection 1.3 for a comprehensive QoS-driven resource management scheme. Previous work does, however, provide techniques for dealing with the specific problems of providing QoS in the network, or CPU, or I/O subsystem.*

The goal of the GRM project was to define an architectural framework for performing comprehensive QoS-driven resource management.† Since a significant amount of previous work had focused on specific aspects of the problem, we took a top-down approach, attempting to understand QoS and its relationship to resource management at its most fundamental level, to the definition of a framework into which the various results of previous and future work could fit. The goal of the GRM project was to develop, in the abstract, a comprehensive QoS-driven resource management solution, so that future work in specific areas of the problem could be coordinated to converge to a complete solution.

We attempted to build on the work of other researchers by abstracting their specific results and addressing areas that had not been investigated by the research community. Our emphasis was on developing a high-level framework and not on the details of any of the specific components that fall within the framework.

1.6 POTENTIAL APPLICATIONS OF QOS-DRIVEN RESOURCE MANAGEMENT

Our QoS-driven resource management framework can be applied to any system where QoS-adaptive applications share (compete for) scarce system resources. All of the applications in the system need not be QoS adaptive, or even QoS aware. For instance, hard real time applications can be allocated the resources that they need as part of the design process, so that they are in fact not competing for the resources that they use. Also, applications that do not have any QoS constraints (i.e., best-effort applications) can run in the system, using the resources that are not allocated to the QoS-aware applications. Following is a short list of representative application areas.

- C⁴I systems—The need for QoS-driven resource management is perhaps clearest in military C⁴I systems. Substantial portions of these systems run in hostile, unpredictable environments. There is a clear definition of priority and of the relative importance of the individual applications that run within these systems. Many of the

*CPU: central processing unit; I/O: input/output.

†In the remainder of this report, we use the terms framework and architecture interchangeably.

applications, such as videoconferencing or review of surveillance video, are (or can be made) QoS adaptive. Different types of users can have very different QoS requirements for similar applications. For instance, soldiers at the front lines, when viewing video of the enemy's positions, may have very strict time requirements and may be willing to sacrifice the quality of the video for its timeliness, while military planners involved in a collaborative planning session may be willing to postpone their activity in order to be assured high-quality videoconferencing.

- **Survivable systems**—For a system to be survivable, it must contain proactive and reactive mechanisms to ensure that the critical functions performed by the system continue to be performed, regardless of the state of the system. One of the key reactive mechanisms is resource allocation, to make certain that the available resources are used to perform the most critical tasks. In its simplest form, resource allocation can be accomplished by means of a simple prioritization scheme. However, if the critical tasks are QoS adaptive, QoS-driven resource management can provide a more flexible and efficient solution.
- **Hospital computing system**—One can picture a hospital of the future, where a common computing infrastructure supports applications ranging from telesurgery and telemedicine to patient monitoring, video on demand to patients' rooms, videoconferencing for patients and their families, and e-mail and World Wide Web (Web) browsing for hospital staff and patients. In order to efficiently share a common infrastructure, these diverse applications require resource management support. Since many of the applications are QoS adaptive, QoS-driven resource management is a natural choice for this type of system.
- **Next-Generation Internet (NGI)**—The current Internet is, as the name implies, a network of networks. Since it provides no QoS assurances to the applications that use it, it is very difficult for applications such as Internet phone or teleseminar to provide any QoS assurances to their users. If the NGI is to provide such QoS assurances, it will need to perform QoS-driven resource management; hence, it is a potential application of our architecture. We believe that the NGI should be viewed not simply as a network of networks, but as a system of systems, with complete end-to-end, rather than simple networking services available from Internet service providers.

1.7 ADAPTIVE QOS-DRIVEN RESOURCE MANAGEMENT ARCHITECTURE

QoS-driven resource management can be described as the process of allocating shared system resources to applications in such a way that the QoS experienced by the users is acceptable to them. Three groups of decision makers can participate in this process, each possessing its own perspective on the problem. The first of these is the users and applications whose perspective we call the *application perspective*. The goal of each user is to derive benefit from the system by running applications. For each application, the user quantifies benefit in terms of the QoS provided by the application. For most applications, the user can receive benefit from the application for more than one set of QoS values. Thus, the user is willing to accept lower-than-nominal QoS values and can make trade-offs between the degradation of the individual QoS parameter values. The users do not necessarily know or care about each other and are oblivious to the impact of their desires on

each other. Each application can be described in terms of its resource demand as a function of the QoS provided to the user. Applications may be able to perform their functions in different ways, providing the same QoS through different combinations of resource demand (e.g., a more communication-intensive approach versus a more computation-intensive approach).

The second perspective is the *resource perspective*. Individual resources understand only the work (pieces of applications) that they perform. Different resources may have their own policies concerning their use (e.g., scheduling or security policies).

Finally, the third perspective is the *system perspective*. The system perspective is the place where the resource and application perspectives converge. The different applications, each with their individual QoS goals, compete for the use of shared resources, each of which can have its own scheduling policies. The system maintains policies for reconciling the potentially conflicting goals of individual applications and resources. It can make decisions concerning which “common services” to employ on behalf of an application (e.g., the Oracle Corporation [Oracle] database or the Informix Software, Inc. [Informix] database) and decisions concerning which resources (and how much of them) to allocate to each application. In making these decisions, the system is driven by a set of objectives, such as to maximize the benefit received by the application users, the total amount of work that is performed, or the number of users that are served.

The goal of our QoS-driven resource management architecture is to serve as a system-wide framework within which the various resource management decisions discussed in Subsection 1.8 can be performed in a coordinated fashion. We believe that a proactive, system-wide entity, which we call the *resource manager*, must implement the system policies, coordinating the application-, resource-, and system-level resource management decisions.* The performance and objectives of this system should be expressed in terms of application-level QoS, and a common definition of QoS should serve as the glue that unifies the various pieces.

Our architecture employs an information-driven approach in providing such a framework. We have defined a QoS taxonomy and a set of models, which capture the information from each of the three perspectives.

1.7.1 QoS Taxonomy

In order to design a comprehensive QoS-driven resource management system where heterogeneous applications compete for heterogeneous system resources, we have developed a QoS taxonomy that captures, in a generic fashion, the various facets of QoS [Sabata et al. 1997]. The primary categories within this taxonomy are metrics and policies. Metrics specify quantifiable QoS parameters, and can be further grouped into the following classifications: *performance specifications*, *security levels*, and *relative importance*.

Performance metrics specify the parameters related to the performance of a task. For example, end-to-end delay, total volume of computations, and bit error rate are performance measures. Performance QoS is defined in terms of *timeliness*, *precision*, and *accuracy*.

Security levels define the data security level that must be provided to the applications. *Relative importance* represents the price (cost) that the user is willing to pay for a service of a given quality (in a system where the users compete for resources), or is a measure of the importance of the work (in a system of cooperating users).

*The resource manager need not be implemented as a centralized entity.

QoS policies are divided into *level of service* and *management policies*. *Level of service* is defined as the type of QoS commitment given to the application. *QoS management policies* define the actions to be taken by the system under different situations. For example, in case of an unforeseen scarcity of a resource, the application may be willing to go through a renegotiation and accept a lower quality of service instead of being denied the service. Management policies also describe the nature of the interactions between the applications and the system. In Section 2 we describe our QoS taxonomy in detail.

1.7.2 Resource Model

The *Resource Model* (RM) captures information about individual system resources. Resources represent the smallest groupings of hardware (and supporting software) over which the resource manager has direct control. We intentionally do not specify the precise granularity of the resources, so as to allow the system designer the flexibility to determine the level of granularity at which QoS-driven resource management is performed.

The RM captures and abstracts resource-specific information that is required by the resource management algorithms. Examples of resource attributes that appear in the RM are resource types, performance characteristics, and scheduling policies. In Section 3 we describe our RM in detail.

1.7.3 System Model

The *System Model* (SM) describes the layout of the system resources and imposes a management structure onto the system resources. A management structure is required for the definition of independent management domains (groupings of resources that are managed under the same set of policies).

We model distributed systems by using a subsystem-resource hierarchical structure. Resources form the bottom layer. A set of resources governed by a single resource management scheme form a *subsystem*. A set of subsystems governed by a single resource management scheme form a higher-level subsystem, which we call the *parent subsystem*. This hierarchical structure continues until the complete system is defined.

The parent subsystem sees all of the resources and child subsystems within it as black boxes whose internal composition is hidden. Each subsystem has a *QoS interface* that enables the parent subsystem manager to communicate QoS information with each child subsystem manager. Two main types of communications occur over these interfaces: (1) parent subsystems pass service requests to their children and negotiate the achievable QoS; and (2) child subsystems pass aggregate resource usage statistics to their parents. This hierarchical representation enables us to model heterogeneous systems running different network protocols, operating systems, and resource management schemes. The details of the SM are presented in Subsection 3.4.

1.7.4 Application Model

Information from the application perspective is captured by three abstractions: the *Logical Application Stream Model* (LASM), the *Application Invocation Model* (AIM), and the *System-Specific Application Stream Model* (SASM). These abstractions capture information about the characteristics of heterogeneous applications, so that the resource manager can make resource management decisions. The application model is used to describe both application- and system-level software. If we picture the system hardware and software as a stack, with the hardware

forming the bottom layer and the application software forming the top layer, a horizontal line can be drawn to divide the software into (1) the software that is modeled via the application model and is thus managed by the resource manager, and (2) the software that is considered to be part of the resources. We leave it to the system designer to determine where this line should be drawn.

The LASM captures the structure of an application in a system-independent manner (i.e., without referencing a specific RM or SM). The AIM contains the information that the user (or the application on behalf of the user) supplies when an application is invoked. Finally, the SASM represents a specific invocation of an application (described by a LASM) on a specific system (described by an SM). The details of the three application models are presented in Section 3.

1.7.5 Resource Manager Structure

Figure 1 shows the relationships between the system, resource, and application models. The information in these models is used at run time by the resource management algorithms to make resource management decisions. The SMs and RMs describe the infrastructure, while the LASM describes the applications. These models (the SM, RM, and LASM) are fairly static, changing when resources are added or removed, new applications or versions of applications are introduced into the system, and old applications or versions of applications are retired from the system. Requests to perform an application are described as entities in the AIM. This model is dynamic, changing with the arrival and departure of application requests. The QoS resource manager uses

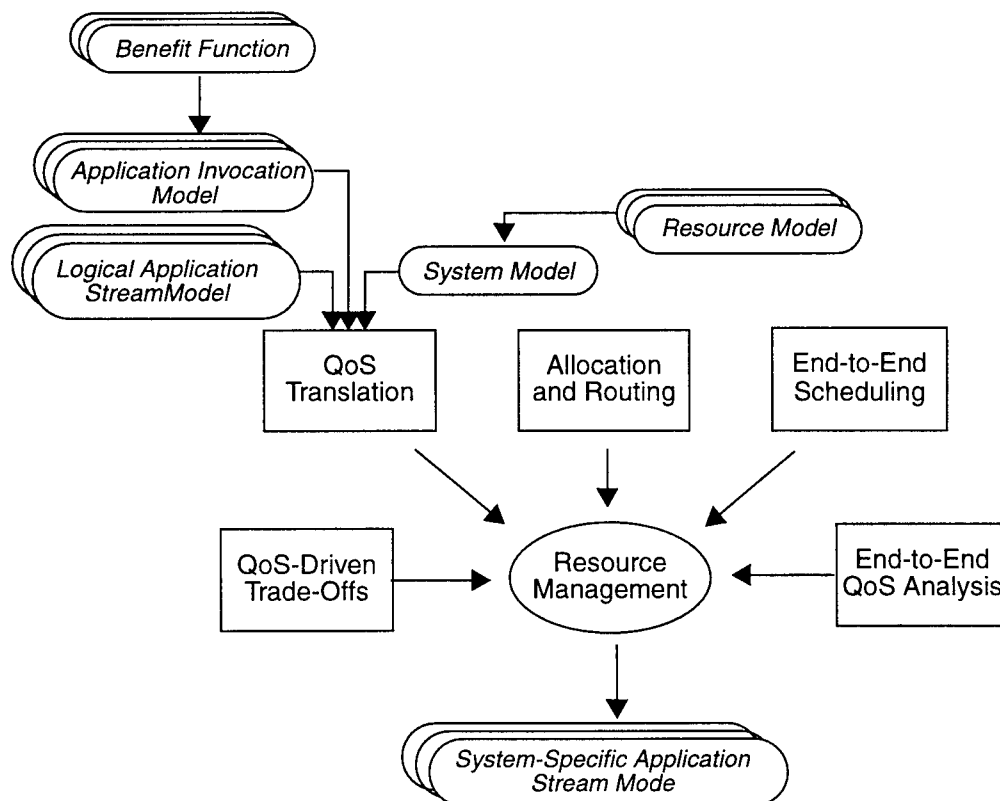


Figure 1. Resource Management Architecture

the information in these models to build the SASM for each active application instance. The SASMs for all application instances represent all of the QoS-controlled work in the system. This model (the SASM) is very dynamic. It changes with the arrival and departure of application requests, as well as with changes in resource availability that result from changes in the state of the resources.

This information-driven approach allows the system designer to define algorithms that “simultaneously” make the codependent application-, resource-, and system-level resource management decisions. We feel that this approach has several benefits: (1) since the application-, resource- and system-level decisions are codependent, it is more likely that a coherent resource management policy will be achieved if they are all made by one entity (centralized or distributed); (2) this approach makes it easier to enforce system-wide policies; and (3) this approach requires the development of less application-specific management code, because the bulk of the management code will be contained in the common resource manager.

There are several major resource management algorithms. *QoS translation* translates the application-level QoS requirements into middleware, operating system, and network-level QoS requirements. *Allocation and routing* algorithms determine which resources to allocate to each application. Since multiple applications execute on each resource, *end-to-end scheduling* algorithms determine, on the basis of QoS requirements, when each application is eligible to use each resource. *End-to-end QoS analysis* algorithms determine whether an allocated and scheduled application will indeed achieve its desired end-to-end QoS; and *QoS-driven trade-offs* are performed when the desired QoS can be achieved. Although we list these as separate algorithms, they are closely interrelated and must be developed as a set. These algorithms are described in more detail in Section 4.

1.8 RESOURCE MANAGEMENT WITHIN A DISTRIBUTED SYSTEM ARCHITECTURE

Adaptive QoS-driven resource management involves all layers of the distributed system, which may comprise heterogeneous components, span multiple administrative domains, and support heterogeneous applications with heterogeneous QoS requirements. We therefore believe that this problem must be solved within the system architecture. This architecture should be scalable, in terms of both the physical size of the system and the level of granularity at which the lowest-level resource management decisions are made.

An architectural solution provides a number of benefits:

- It allows for the coordination of the various decisions that together constitute QoS-driven resource management.
- It provides an orderly and scalable approach to the management of heterogeneous resources.
- It provides an orderly approach to the management of heterogeneous applications that compete for system resources.
- It allows the operations that are common to all applications or resources to be identified and implemented in a common way (e.g., in middleware).

Since resource management is not the only architectural consideration in a distributed system, we examine its relationship to other system-wide operations. We argued in Subsection 1.3 that QoS-driven resource management is required to allow applications to provide QoS assurances, despite the unpredictable nature of distributed systems. Since security attacks and failures are events whose occurrence cannot be predicted, a QoS-driven resource management architecture cannot be developed independently of the security and fault tolerance architectures. Since considerable research has been done in the areas of fault tolerance and security, our goal has been to identify the relationship between those areas and resource management, and to structure our QoS-driven resource management architecture so that it can fit into a system architecture that addresses all three. For the sake of brevity we do not discuss this relationship in detail in this report. Sydir et al. [1997] describe this relationship in more detail.

In Figure 2 we present a flow chart that delineates the high-level components of an adaptive resource management system within the broader context of a distributed system. We briefly describe each of these components, beginning with monitoring. The resource manager employs *system monitoring* mechanisms that alert the resource manager to changes in system state. The resource manager invokes different algorithms specific to the type of system state change, each of which is described below.

When a new application requires QoS support, or an existing application requests a QoS upgrade, *allocation and scheduling* algorithms are invoked to determine which resources should be used to support the application. Allocation and scheduling algorithms are influenced by *proactive fault handling* algorithms that replicate application components, based on statistical information

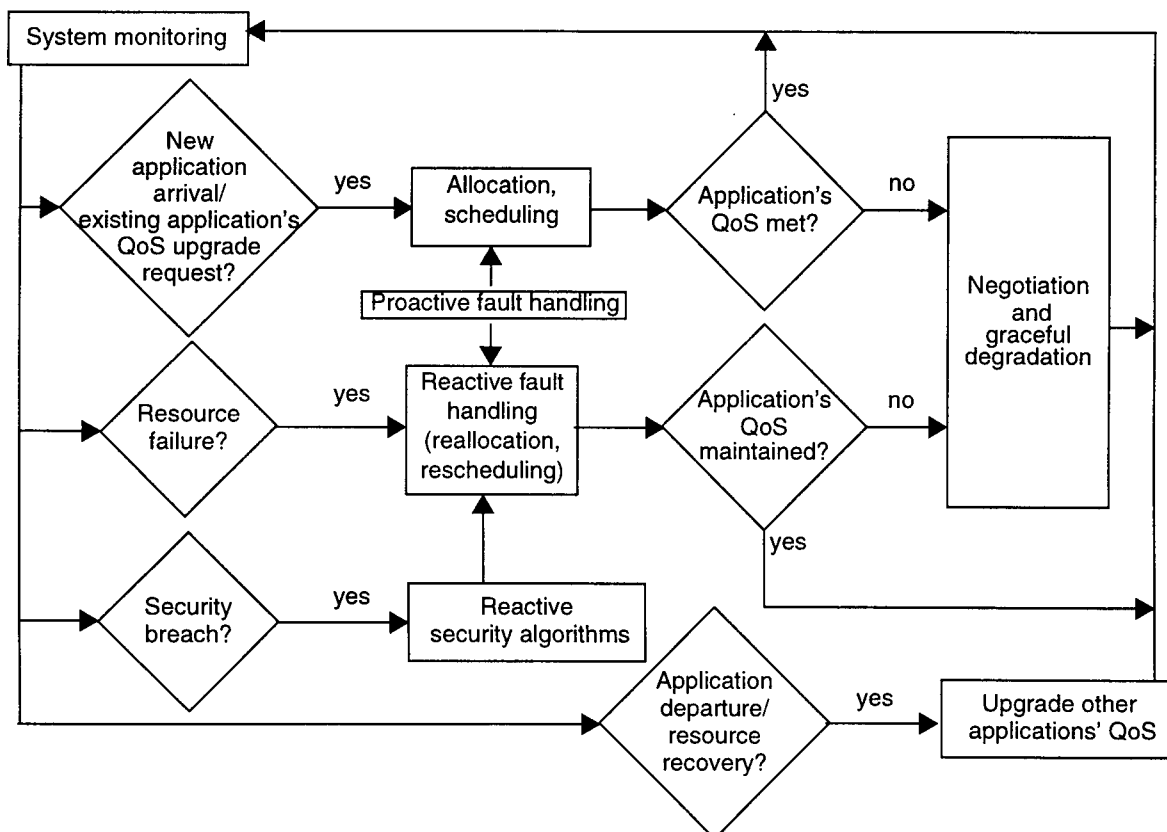


Figure 2. Adaptive Resource Management

about future resource failures and application QoS needs. If necessary, the allocation and scheduling algorithms reallocate or reschedule existing applications without affecting their QoS, in an attempt to support the new request. If this is not possible, *QoS negotiation* algorithms must be utilized to find a level of QoS that can be provided to this new application. On the other hand, if the new application is deemed critically important, system policy may dictate that the resource management scheme provide the necessary QoS to this application by *gracefully degrading* the QoS of other existing applications.

A resource failure causes the invocation of *reactive fault handling* algorithms, which attempt to preserve application QoS by reallocating and rescheduling applications to mask the loss of the resource. If application QoS cannot be maintained, then the *graceful degradation* algorithm degrades the QoS of applications in order to preserve the operation of highly important applications, with minimal adverse impact upon less important applications.

If a security break occurs, *reactive security* algorithms are utilized to isolate the resources under attack. From a resource management perspective, these compromised resources are equivalent to failed resources, so that reactive fault handling algorithms are invoked to reallocate and reschedule applications.

Finally, resource recoveries, the cessation of security attacks, the departures of applications, and voluntary reductions of their QoS requirements enable other applications' QoS levels to be increased, if the applications so desire. In such cases, the resource manager must make intelligent QoS trade-offs to maximize the benefit to each application.

2 TAXONOMY FOR QOS SPECIFICATIONS

2.1 INTRODUCTION

QoS parameters are expressed in terms of different units at different layers of the system. Application QoS parameters are a function of an application's goals and design. Resource-level QoS parameters depend on the design of the resources and on their control parameters. Applications running in the system must be able to specify their QoS requirements and should be able to operate over a range of QoS values. The system, in turn, must be aware of application QoS parameters and must be able to translate them to resource-level QoS parameters. For example, the communication resources understand QoS parameters such as packet delay and packet jitter, while a video on demand application understands QoS parameters such as frame delay and frame jitter. Although translation between the two delay parameters is fairly simple, translation between the two jitter parameters is not.

A distributed system that provides QoS guarantees to general applications, sharing general system resources, must be based on a QoS framework. Such a framework consists of a QoS specification taxonomy, and a QoS architecture that integrates the different components in the various layers of the system. A general taxonomy leads to a better understanding of the QoS parameters and their interrelationships. The final goal is to have generic translation schemes that can be the QoS interfaces between the various system components. The QoS-based framework will help in identifying the functional requirements for the management and monitoring mechanisms of the distributed system.

In this section we define and classify the different QoS attributes and place them into a general QoS taxonomy. In Subsection 2.2 we present a brief overview of past work, followed by descriptions of the taxonomy in Subsection 2.3, performance metrics in Subsection 2.4, security in Subsection 2.5, relative importance in Subsection 2.6, and policies in Subsection 2.7; and finally some conclusions in Subsection 2.8.

2.2 BACKGROUND

Although there has been work in QoS-driven resource management, most of this work has focused on either the network or operating system layer [Campbell, Coulson, and Garcia 1993; Katcher, Arakawa, and Strosnider 1992; Nahrstedt and Steinmetz 1995; Stewart, Schmitz, and Khosla 1989; Verma, Zhang, and Ferrari 1991; Vogel et al. 1995]. Resource management at each of these layers has been done separately, with very little understanding of how the confluence of resource management at the different layers can provide end-to-end QoS support to applications. As a result, these disjoint approaches to resource management are not sufficient for providing end-to-end application QoS support. Refer to Sydir et al. [1997b] for a thorough description of previous work in each of these layers.

Campbell, Aurrecoechea, and Hauw [1996] were among the first to recognize the need for an integrated approach to resource management. They presented an integrated framework that deals with end-to-end application QoS requirements. The notion of flow is introduced as an important abstraction within the framework. Flow is defined to characterize the production, transmission, and consumption of the data associated with a single medium. Flows are either unicast or multicast and generally require end-to-end admission control.

Campbell, Aurrecochea, and Hauw [1996] define QoS, based on the notion of flow, to include specifications for *flow synchronization*, *flow performance*, *level of service*, *QoS management policy*, and *cost of service*. This taxonomy is the best we have seen in the literature; however, it fails to include important concepts such as the precision of the data produced, and application security and fault handling requirements. The focus of all work in the area of QoS has been on the timing and error aspects (timeliness and accuracy) of systems. The volume of work needed to perform a service has not been considered to be a dynamic adjustable parameter. A QoS-based system, however, should be able to dynamically adjust the amount of work performed (e.g., by using hierarchical encoding of video data, or by resizing the video frame). This capability would allow the system to make trade-offs between the various QoS parameters, when sufficient resources are not available.

2.3 QOS TAXONOMY

In order to design a system where multiple applications coexist within a QoS management framework, the applications must either have a common understanding of how QoS should be specified, or must be able to map their individual specifications into a common one. To this end, we define in a generic fashion the various facets of QoS. We believe that QoS parameters are grouped into metrics and policies. The metrics measure quantifiable QoS attributes in the applications, system, and the resources. The metrics are further divided into performance metrics, security levels, and the relative importance. The policies describe the system behavior specifications. The primary policies are the management policies, and the levels of service.

We classify QoS parameters according to the taxonomy shown in Figure 3. The primary categories, as noted above, are metrics and policies. Metrics specify quantifiable QoS parameters. Metrics can be further grouped into the following classifications: *performance specifications*, *security levels*, and *relative importance*. Policies are divided into *level of service*, and *management policies*.

Performance QoS is defined in terms of *timeliness*, *precision*, and *accuracy*. Performance metrics specify the parameters related to the performance of a task. For example, end-to-end delay, total volume of computations, and bit error rate are performance measures. For each performance parameter there are absolute specifications and consistency specifications. Consistency

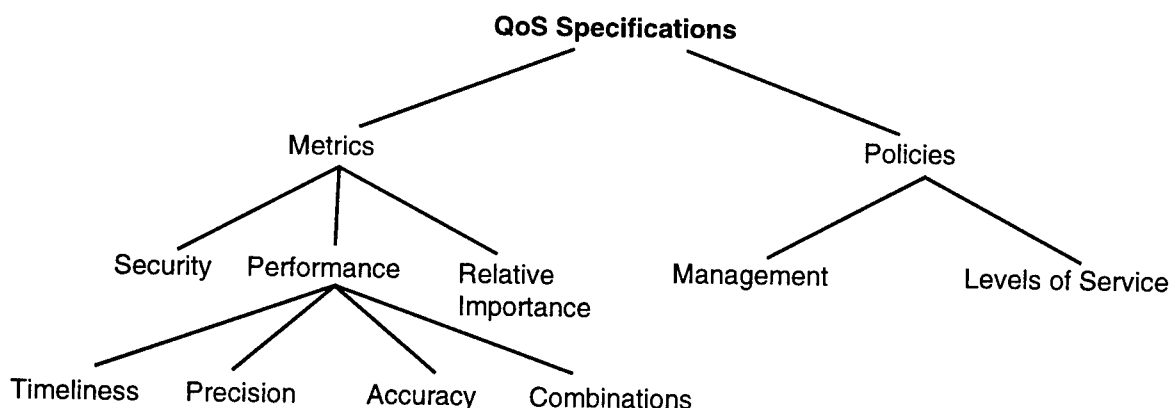


Figure 3. A Taxonomy for QoS Specifications

specifications define the relations between the different task flows and the different instances of a task flow. For example, jitter and synchronization metrics measure the consistency of the delay parameter between different instances of a task, and between different task flows, respectively.

Relative importance represents the price (cost) that the user is willing to pay for a service of a given quality (in a system where the users compete for resources), or a measure of the importance of the work (in a system of cooperating users). *Security levels* define the data security level that must be provided to the applications [Sydir et al. 1997a].

Level of service is defined as the type of QoS commitment given to the application. QoS *management policies* define the actions to be taken by the system under different situations. For example, in case of an unforeseen scarcity of resource, the application may be willing to go through a renegotiation and accept a lower QoS instead of being denied the service. Management policies also describe the nature of the interactions between the applications and the system.

2.4 PERFORMANCE METRICS

2.4.1 Timeliness

Timeliness parameters define a class of metrics that measure time related entities. Timeliness is expressed in units of time. Its definition is straightforward, because time is understood in the same way by humans and computers, and has roughly the same meaning in all layers of the system, from the applications down to the network. We have defined timeliness as a representation of the timing requirements for performing a given piece of work.

Timeliness parameters are metrics that measure

1. The total time taken to complete (from beginning to end) a task (a unit of work [UoW] or service). This is measured as the delay, or latency, or time to complete.
2. The start time (earliest/latest) for a task.
3. The deadline (earliest/latest) for the completion of the task.
4. The variability in time to complete a task (jitter). This parameter measures the internal consistency of the timeliness parameters.
5. The relationship between the deadlines and the start times of the different tasks (synchronization). This parameter measures the mutual consistency of the timeliness parameters.
6. The statistical distribution of each one of the above parameters.

The first three items are absolute specifications, Items 4 and 5 are consistency specifications, and Item 6 applies to all the parameters. The absolute metrics are used by schedulers to schedule the task at the appropriate time. The consistency specifications require the introduction of mechanisms that ensure that the time relationships between the different task flows and within a task flow are maintained. The statistical distribution of each parameter considers it to be a random variable and describes the distribution of the variable. This distribution allows the tolerance associated with each parameter to be specified. Also, in the case of repeating tasks the statistical distribution describes how the parameters change over the different instances of the task.

2.4.2 Precision

Precision parameters specify volume-related quantities. Precision and accuracy require more detailed definition, since the same data can be viewed in different ways (and can be understood in different ways) by different components of the system. Since precision and accuracy are attributes of the data that flows through an application, it is important to distinguish between the content of the data in and the series of bits that represent the data. We define data content to be the meaning of the data. For example, the fact that a floating-point number represents a median of some number of data points is its data content. Since a piece of data that is manipulated by a computer system must be represented as a series of bits, we define data representation to be the computer representation of a given piece of data. Thus, a given piece of data that is understood in terms of its data content can be represented by one or more data representations that differ in format and size. Since a data representation is a tangible construct, it has a spatial characteristic, which we refer to as the volume of data. Volume of data is an amount of data expressed as a number of bits or bytes. It is important to note that the volume of data in a data representation of a given data content depends on the algorithm used to encode the data. The same data content can have different representations whose volumes are different. For example, a floating-point number can be represented in a C program as a float variable or as an ASCII character string.

We recognize that the amount of work that a computer system has to do to transfer or store data is proportional to the size of the data. Thus, the volume of data is a measure of the amount of work required to transfer or store it. Analogously, we define the volume of computational work as the total amount of computations (e.g., FLOPS^{*}) to complete the task. “Work” is used here in a very broad sense, to mean the usage of all involved hardware resources. As in the case of data content, a given computation executes a function that can be implemented in one of many possible ways. Each implementation performs a different volume of computation to do the same amount of “work content.”

We can now define precision as it applies to content and representation. In general, the precision of representation describes the amount of data or work. The precision of content is defined in terms of the specific data content or the functional transform. For example, the precision of the median value for some set of data points can be defined in terms of the number of decimal places to which it is calculated. The precision (or volume) of the representation is defined in terms of its volume (the physical amount of data in bits or bytes). The precision of content in one layer translates into a precision of representation in the next layer, e.g., a data packet in the transport layer is composed of the payload and the header; this content of the data structure is understood by the transport system. But the lower network layer recognizes the data only as a collection of data bytes and represents that as a series of bits. The units of volume can change for the different components of the system, e.g., the number of frames of video in the application translates to the number of bytes of data in the middleware and the network layers.

As we did with timeliness parameters, we can capture the consistency requirements of precision parameters—in this case, by defining jitter and synchronization. The variation in the volume in successive instances of tasks (UoWs or services) measures the consistency of the precision metric. For example, when video is compressed, the application can choose to have the same number of bytes for each frame or may vary the total amount of data. This variation in the total volume of data between frames is captured in the internal consistency of the precision metric.

^{*}FLOPS: Floating point operations per second.

The relationship between the precision of different flows is measured in terms of the consistency between the precision parameters of each flow. This relationship may be important, because the results from two different computations can be combined, and the precision of the data generated in both flows must be compatible for those results to be combined. The precision parameters of interest are

1. The precision of content, for input and output data
2. The precision of representation, for input and output data
3. Internal consistency of precision over a flow (precision jitter)
4. Mutual consistency of precision between flows (precision synchronization)
5. The statistical distribution of the above parameters.

Precision parameters 1 and 2 are absolute specifications; 3 and 4 define the consistency parameters; and 5 defines the statistical distribution of each precision parameter.

2.4.3 Accuracy

Accuracy measures the errors introduced into the data by UoWs and services. The accuracy of data content is also defined in terms of the specific data content. For example, the precision of the data specifies the number of decimal places to which the median of a number of data points is calculated, while the accuracy of the data specifies how many of those decimal places actually contain correct (accurate) data. Finally, the accuracy of data representation is defined as the amount of data volume that is actually correct. (This amount is most naturally specified as a percentage.) In both cases—data content and data representation—accuracy is bounded by precision (e.g., a floating-point number calculated to a precision of three decimal places can be accurate to at most three decimal places). The accuracy of computations is described in terms of the accuracy of the data generated by the computations. The accuracy parameters of interest are

1. The accuracy of content for input and output data
2. The accuracy of representation for input and output data
3. Statistical distribution of accuracy.

Conceptually, we can define an accuracy jitter and synchronization that correspond to the internal and mutual consistency parameters. In practice, however, there seems to be no application of such a concept.

2.4.4 Combination

The three classifications, timeliness, precision, and accuracy, do not represent independent (orthogonal) axes. Since the timeliness, precision, accuracy components of a QoS specification must be provided to the application “simultaneously,” there are cases when QoS can be specified by a parameter that is a combination of these components. Throughput, defined as precision over time, is one of these parameters. At this time, we are not aware of any other practical QoS parameters that are combinations.

2.5 SECURITY

Security metrics deal specifically with policies and mechanisms related to the data security that applications require [Sydir et al. 1997a]. We have defined two security parameters, level of confidentiality and level of integrity. Confidentiality ensures that information does not get into the wrong hands. Integrity ensures that information is and remains accurate. That is, the persons and processes that are allowed to modify a given piece of information are restricted to those that are trusted to do so. Integrity also applies to system components, in that it ensures that they are not modified or replaced (presumably, in a way that violates one or more aspects of security policy). These two QoS parameters express the sensitivity of the data being handled with respect to its confidentiality and integrity. The units in terms of which these parameters are specified are specific to the type of system. For example, in military systems these units might be levels of military security (e.g., unclassified, SECRET).

Availability is commonly used as a third security parameter; it indicates the degree to which sufficient computing resources are available to perform the required work at the desired time. Since the provision of an availability guarantee involves not only security, but also fault tolerance and resource management, we place the availability parameter in the levels of service category (discussed in Subsection 2.7.1).

2.6 RELATIVE IMPORTANCE

In order to allocate a resource among multiple, competing applications, the resource management system requires a way to evaluate the relative importance of the different applications that are contending for the resource.

In the case of competing users (as in commercial systems), the price (cost) that the user is willing to pay for a service of a given quality is an effective mechanism for determining application priority. In the case of cooperating users (as in military systems), the importance of the user and the application can be absolutely defined and used to gauge the relative importance of the work. We also define resource cost functions to express a resource's willingness to provide a given QoS setting. If a given type of resource is scarce, the cost of using that resource increases.

2.7 POLICIES

2.7.1 Levels of Service

An application requires an assurance of the system's level of commitment to providing their QoS needs. This level of commitment dictates the resource manager's choice of a policy to provide the service to the application. This policy can range from a best-effort policy (no guarantees) to one that provides a very high level of assurance that application QoS will be maintained at all costs. We define level of service to be the level of commitment for a task. A service is either a guaranteed service or a best-effort service. The distinction between the two is that the system may provide no benefit to the user of a best-effort service, while the user of a guaranteed service is promised a given level of benefit. Level of service is a metaspecification of the QoS parameters. It provides a policy statement about the way each performance parameter must be monitored and manipulated.

Different levels of guaranteed service can be provided by the system. For example, missing even one QoS requirement can lead to a catastrophic failure for many control and defense applications. On the other hand, catastrophes do not usually result from missing an audio or video application's QoS requirements. We use the availability QoS metric to enable the application to specify the level of guaranteed service it requires. Availability is expressed as the probability that the QoS assurances will be met; thus, it is a meta-attribute for the other QoS metrics.

2.7.2 Management Policies

QoS management policies define application-specific actions to be taken by the resource manager in various situations. For example, in the case of an unforeseen scarcity of resources, the application may be willing to go through a renegotiation and accept a lower QoS instead of being denied the service. There are no quantifiable metrics that describe these policies, but in general, the policies can be classified into different classes of management functions. The user may specify such a class as part of the application requirements. For example, the classes might be "renegotiation allowed" and "renegotiation not allowed." This again is a meta-attribute for the other QoS metrics.

2.8 CONCLUSIONS

In this section we have presented a taxonomy of QoS specifications that is the starting point for defining the QoS interfaces between the different system components. The key to understanding QoS-based distributed systems is recognizing the relationships between the different ways that QoS concepts are defined by different system components. Our taxonomy clearly identifies the different classes of QoS parameters. We see that the fundamental metrics are those of performance, cost, and security. The performance metrics in turn can be classified in terms of timeliness, precision, and accuracy. The concept of these three primary classes of performance is important, because most of the QoS specifications we have seen are merely statements of one or two of these classes. Within each class of the QoS specification there are requirements for absolute quantities and consistency measures, which gives a large, dynamic range of parameter specifications. Finally, the notion of a statistical distribution of each parameter allows the specification of parameter tolerance and variability. The classification of QoS metrics into their most basic groupings provides insights into the common mechanisms for translating QoS specifications between system layers.

3 APPLICATION, RESOURCE, AND SYSTEM MODELS

Adaptive resource management must simultaneously consider the objectives and constraints of applications, resources, and the system. Recall from Section 1 the three perspectives (application, resource, and system) that the system-wide resource management scheme must consider when making decisions.

From the application perspective, applications selfishly want access to enough system resources to achieve the desired level of QoS; they are not concerned about how this access is provided or how it affects other applications. Similarly, each resource (such as processors, disks or communication networks) is concerned only about the set of applications running on it; it does not care about other resources or applications running on other resources. The system perspective, however, has the global view of the applications and the resources. The objectives of the individual applications and the individual resources are likely to be in conflict; thus, the role of a system-wide resource management scheme is to resolve these conflicts. To resolve conflicts, the system perspective captures system policies, including end-to-end scheduling policies, policies to decide which application's QoS to degrade when resources are insufficient to provide the desired QoS to all applications, and admissions control policies.

The three perspectives are captured in our architecture by means of a set of models. Figure 4 shows these different models, and their relationships to each other. Each Resource Model (RM) captures the attributes and scheduling behavior of a single resource, abstracting away the implementation details about the resource while exporting a uniform QoS interface to the system. This "black box" approach enables us to provide end-to-end resource management over heterogeneous computing, storage, and communication resources. The System Model (SM) captures the set of resources that constitute the system, the topological and administrative structure of the system, and the policies and objectives governing each of these administrative domains.

The Logical Application Stream Model (LASM) captures an application's structure, resource requirements, and relevant QoS parameters. The LASM is independent of any system attributes or a user's QoS requirements. Therefore, a LASM can be used by multiple users who wish to invoke

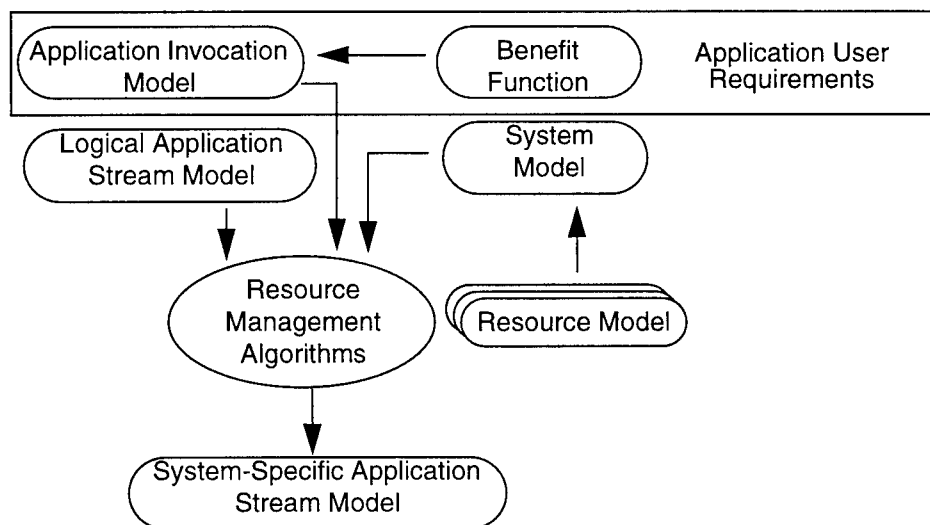


Figure 4. QoS Architecture for Resource Management

the same application with different QoS requirements. The Application Invocation Model (AIM) captures each user's specific QoS requirements. The Benefit Function (BF) captures the user's QoS preferences.

When a user wishes to execute an application, he or she chooses the appropriate LASM and specifies his or her QoS requirements and preferences, using the AIM and the BF, respectively. The system-wide resource management scheme then determines which system resources to use for the particular application and how to schedule this application on these resources, for this particular invocation and with these particular QoS requirements. This information is captured via the System-specific Application Stream Model (SASM).

In the remainder of this section, we describe these models. In Subsection 3.1, we first describe related work. In Subsection 3.2, we describe the application models (the LASM, AIM, BF, and SASM). In Subsection 3.3, we describe the RM. In Subsection 3.4, we describe the SM. In Subsections 3.5 and 3.6, respectively, we describe the subsystem interfaces, and show how the models can be used for resource management.

3.1 RELATED WORK

Several models attempt to capture application characteristics to facilitate resource management [Beck and Siewiorek 1994; Coolahan and Roussopoulos 1983; Gilles and Liu 1993; Ramamritham 1989; Hull and Liu 1993]. These models view an application as a set of subtasks running on a set of ideal resources with identical behavior, and no overhead or blocking components. This view, however, leads to the creation of simple models that unfortunately cannot be used in real distributed systems comprising heterogeneous resources. Our application models, on the other hand, are specifically targeted for modeling distributed applications that execute on heterogeneous computing, storage, and communication resources and have end-to-end QoS requirements.

Some recent work has examined application specifications for graceful degradation. The approach used in imprecise computations [Hull and Liu 1993] is to decompose a task into mandatory and optional components. The resource management scheme must schedule the mandatory components, whereas the resource management scheme attempts to schedule the optional components; the more optional components it can schedule, the more precise will be the results. Our approach is much more general than that of Hull and Liu because we allow QoS trade-offs between a range of QoS metrics, rather than limiting trade-offs to precision and time.

Another approach [Parris, Ventre, and Zhang 1993] allows the user to specify a range of acceptable values for each QoS metric; however, all QoS levels within this range are considered to provide equivalent benefits to the user. Initially, when resources are plentiful, the resource management scheme maximizes the QoS provided to each application. Later, when a resource shortage occurs, all applications have their QoS reduced by equal amounts, down to their minimum QoS level.

Our approach, which extends that of Davis, Downing, and Lawrence [1994], is better than that of Parris, Ventre, and Zhang [1993] in three areas. First, their work only applies to modeling network traffic only, whereas ours applies to the modeling of end-to-end applications executing over computing, storage, and communication system resources. Second, our BF is a more

comprehensive abstraction for modeling applications' QoS preferences in order to convey the relative benefits that the users receive if the system provides a level of QoS; Parris, Ventre, and Zhang assume that the benefits are equivalent within the minimum and maximum QoS bounds.

Our new model is an extension of work by Chatterjee and Strosnider [1995] that modeled distributed applications executing over heterogeneous system resources. Our model improves upon the previous one in the following ways: (1) it is more scalable; (2) it enables the dynamic restructuring of applications during resource management; and (3) it enables trade-offs among many application QoS metrics during graceful degradation.

Most previous work on resource models capture only a resource's attributes and not its behavior. For example, most published research on modeling wide-area networks has used link bandwidth, maximum packet size, and topology to model such networks. This approach is adequate to model systems composed of homogeneous resources, since the system-wide resource management scheme can use these resource attributes (link bandwidth, maximum packet size, and topology) to determine whether the resources can support the applications. Extrapolating this approach to the modeling of heterogeneous resources requires the system manager to understand the characteristics of each resource type in detail. This approach is not scalable or practical; for example, the addition of a new resource type to the system requires the system manager to be augmented with knowledge about this new resource type.

Our black-box approach to the development of an RM was originally described by Chatterjee and Strosnider [1995]. Their RM included resource-specific attributes and an scheduling model. Network SMs,^{*} processor scheduling models,[†] bus models,[‡] and a generic disk^{**} were also included. The attributes and the SM were used to calculate resource-type-independent metrics to determine whether applications met timing requirements.

3.2 APPLICATION MODELS

3.2.1 Logical Application Stream Model

In this subsection, we introduce a model (the LASM) that captures a distributed application's (1) structure and (2) relevant QoS parameters, at a *logical* level, irrespective of the user's QoS requirements and of the system on which the application may execute. The LASM is used by the system-wide resource management scheme to structure each end-to-end application, allocate resources to each application, and to schedule these applications on these resources. Later, when the system state changes, the system-wide resource management scheme can use the LASM to reallocate, reschedule, and restructure these applications dynamically.

The concept of and the need for reallocating and rescheduling applications on system resources is well documented and understood; the concept of restructuring applications dynamically is not as well understood. Therefore, in Subsection 3.2.1.1 we first motivate the value

^{*}Strosnider and Marchok 1989; Lee and Shen 1990; Sathaye and Strosnider 1992; Sathaye 1993; Raha 1995.

[†]Jeffay and Stone 1993; Stewart, Schmitz, and Khosla 1989; Tindell, Burns, and Wellings 1992; Katcher, Arakawa, and Strosnider 1992, 1993; Chatterjee and Strosnider 1996; Katcher, Kettler, and Strosnider 1994.

[‡]Kettler, Lehoczky, and Strosnider 1995; Kettler and Strosnider 1994.

^{**}Daigle and Strosnider 1994.

of the system-wide resource management scheme's capability to dynamically restructure applications as the system state changes. Having defined the value of enabling these three forms of adaptivity, we then describe the LASM in the remaining two subsections.

In Subsection 3.2.1.2, we introduce a single-layer Logical Application Stream Model; this model is used by the system-wide resource management scheme to structure each end-to-end application, allocate resources to each application, and schedule these applications on these resources. In Subsection 3.2.1.3, we then introduce a recursive LASM that further enables the system-wide resource management scheme to structure applications, based on the system state, and also facilitates the modeling of large-scale applications. When the system state changes, the system-wide resource management scheme can use the LASM also to reallocate, reschedule, and restructure these applications dynamically.

3.2.1.1 Dynamically Restructuring Applications

Consider the following scenario. Multiple users are running several continuous media applications (i.e., video conferences) at a certain time, t_0 , on a shared system. These applications have QoS assurances from the system-wide resource management scheme that their QoS will not fluctuate as the system load changes. At a later time, t_1 , a new user wants to run a virtual-reality-based airplane simulation on the shared system. This new application places a high load on computing resources.

Assume that insufficient computing resources remain to run the virtual reality application and to provide it the level of QoS it requires. The remainder of this subsection will describe how the system-wide resource management scheme can *dynamically restructure* some of the video applications to free computing resources so that the system can support the new application without compromising other applications' QoS.

As stated earlier, the algorithms comprised in each application, and their execution order, define the structure of the application. Therefore, *restructuring* denotes using different algorithms to achieve the same application end-to-end functionality. Take a video application as an example. A typical video application may comprise the following four algorithms: fetch video, compress video, decompress video, and display video. The use of different compression algorithms would then constitute the restructuring of the video application. In the example above, the system-wide resource management scheme may have utilized MPEG* compression at time t_0 . An MPEG algorithm generally utilizes a very complex compression algorithm to increase the compression ratio; although this algorithm places a high load on computing resources, it significantly reduces the load on communication resources. However, at time t_1 , the algorithm may not leave sufficient computing resources to execute the virtual reality application. Therefore, the system-wide resource management scheme may restructure some of the video applications to use JPEG† compression instead; JPEG uses a less complex compression algorithm, imposing a lesser load on computing resources, but a higher load on communication resources, due to its smaller compression ratio. Therefore, restructuring frequently enables the system-wide resource management scheme to trade the use of one resource for another, increasing the number of ways it can provide QoS to applications.

*MPEG: Motion Picture Experts Group.

†JPEG: Joint Photographic Experts Group.

3.2.1.2 Single-Layer Model

A distributed application can be viewed as a set of tasks executing in a specific order on a set of specific computing, storage, and communication resources. Accordingly, algorithms running on computing resources, data transfer over communication resources, and data transfer to and from storage resources are all denoted *tasks*. We further denote an atomic task as a *Unit of Work*. From a resource management perspective, each UoW within this application transforms the application quality of service. This decomposition of an application into UoWs that transform data and QoS is a fundamental concept of the LASM. UoWs are the smallest components of applications that are used for allocation and routing.

The *LASM* captures application attributes at a *logical*, or system- and user-QoS-independent, level. The structure of a single-layer LASM is represented using a directed graph, where the graph nodes correspond to *Logical Units of Work* (LUoWs) and graph edges, denoted *Logical Edges* (LEs), dictate the order in which the LUoWs execute. We use the *Logical Constraints* (LCs) set to capture application QoS relationships. The QoS parameters that must be assigned at invocation by the user are declared in the *Logical Parameter* (LP) set. Therefore, the LASM is represented in aggregate as follows: $LASM = (LUoW, LE, LC, LP)$. These four attributes, and an example, are described below.

Each LUoW has a set of attributes, some of which are mandatory (must be specified), while others are optional (may be specified). These mandatory attributes are as follows:

- **Id**, *LID*: unique identifier for this LUoW
- **Name**, *LN*: name of work to be done
- **Input and output data type**, *LIDT*, *LODT*: type of data flowing into and out of the LUoW
- **Role**, *LR*: behavior of the LUoW.

Each LUoW may have one or more of the following roles:

- **Producer**: The LUoW produces one set of output data each time it completes execution.
- **Consumer**: The LUoW consumes one set of data each time it executes.
- **Join**: The LUoW consumes multiple sets of data each time it executes.
- **Feedback**: The LUoW consumes feedback data each time it executes.
- **Synchronization**: The LUoW consumes multiple sets of data each time it executes and produces an identical set of data when it completes execution.

Each LUoW also has the following optional attributes, which may or may not be assigned:

- **QoS parameters** (described below)
- **Resource constraint**, *LRC*: constraints on which resources the LUoW may execute
- **Eligibility time**, *LET_i*: time when the *i*th instance of the LUoW is eligible to execute
- **Absolute deadline**, *LAD_i*: time when the *i*th instance of the LUoW must complete execution.

The following QoS parameters are relevant for each LUoW:

- **Input data volume and accuracy**, $LIDV_i$, $LIDA_i$: the number of bytes, and the minimum accuracy of this data, required for the i th instance of the LUoW to execute
- **Output data volume and accuracy**, $LODV_i$, $LODA_i$: the number of bytes, and the accuracy of the data, produced when the i th instance of the LUoW completes execution
- **Data precision and accuracy**, LDP_i , LDA_i : the precision and accuracy of the data, in application-specific terms
- **Rate**, LF : the frequency at which the LUoW executes.

LEs can be classified as either feed-forward or feedback edges. The reason for separating feedback edges from feed-forward edges is to avoid cycles in the application graph and thus simplify allocation, scheduling, and analysis.

The LC set is used to assign all mandatory LUoW attributes, and whichever optional LUoW attributes the application developer desires. Variables should be used in most assignments, since most assignments are functions of the users' QoS requirements. All of these variables must then be declared in the LP set, according to the following convention: {variable name, string, unit, variable type}. Common units include BYTE or SECOND; common variable types include FLOAT or INT. At invocation, the user specifies his or her QoS by defining these variables in the LP set, using the AIM.

Figure 5 illustrates an example videoconference application captured by means of a single-layer LASM. The boxes correspond to LUoWs and the arrows correspond to LEs. Because the LUoWs are not bound to system resources, the exact number of I/O hops cannot be specified at the LASM level; instead, we implicitly specify I/O between LUoWs by using the data edges.

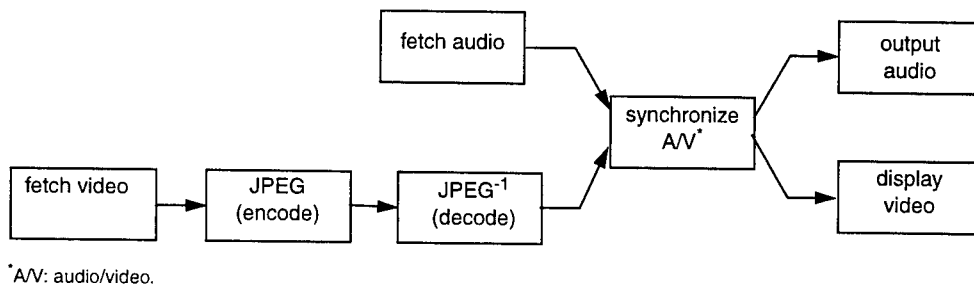


Figure 5. Logical Application Stream Model

LUoW attributes are specified by means of the LC set. A few example LUoW attributes are given in Table 1.

Table 1. LUoW Attributes

| LUoW NAME | FETCH AUDIO | FETCH VIDEO | JPEG | JPEG-1 | SYNC A/V | DISPLAY VIDEO | DISPLAY AUDIO |
|--------------------|-------------|----------------------|-----------------------------|----------------------------|---------------------------------------|----------------------|---------------|
| ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Input Data Type | Ø | Ø | RGB* Video | JPEG | {Audio, RGB Video} | RGB Video | Audio |
| Output Data Type | Audio | RGB Video | JPEG | RGB Video | {Audio, RGB Video} | Ø | Ø |
| Role | {PRODUCER} | {PRODUCER} | {PRODUCER, CONSUMER} | {PRODUCER, CONSUMER} | {SYNC-RONIZATION} | {CONSUMER} | {CONSUMER} |
| Input Data Volume | Ø | Ø | Width × Height × Res | (Width × Height × Res)/ CR | 2*LF† (fetch audio)/ LF (fetch video) | Width × Height × Res | 2 Bytes |
| Output Data Volume | 2 bytes | Width × Height × Res | (Width × Height × Res) / CR | Width × Height × Res | | Ø | Ø |
| Rate | 8 kHz | FR‡ | | | | | |

*RGB: red, green, and blue.

†LF: LU of frequency.

‡FR: frame rate.

An important constraint for interactive video is end-to-end latency. Rather than assigning specific LUoW attributes to achieve this constraint, we instead constrain the time duration between the start and end times for two LUoWs:

$$\forall i, i \geq 0, PAD_i(\text{display video}) - PET_i(\text{fetch video}) \leq VL.$$

Note that frame rate (FR), frame size (Width × Height), frame resolution (Res), video compression ratio (CR), and video latency (VL) are all given via variables. These variables must therefore be declared in the LP set. An example entry for VL is {VL, "Max. End-to-End Latency," SEC, FLOAT}. This entry indicates that at invocation time, the user will be asked to enter a value for the "Max. End-to-End Latency," only float type values will be accepted, the entered value will be considered to be in seconds, and the entered value will be assigned to the variable VL.

3.2.1.3 Recursive Model

The single-layer LASM described above enhances the application's portability and enables reallocation and rescheduling as the system state changes. The recursive LASM described in this section further enables application restructuring and facilitates the modeling of large-scale applications.

The structure of a recursive LASM is also represented by means of a directed graph. Unlike the single layer LASM, graph nodes now correspond to *Logical Services* (LSs). Graph edges, still denoted LEs, now dictate the order in which the LSs execute. Each LS has the same attributes as the LUoW attributes listed in Subsection 3.2.1.2. The LC and LP sets are also identical to before.

Each LS is realized by means of a *Logical Realization of Service* (LROS). An LROS is composed either of a set of child Logical Services, executing in a specific order, or of a single LUoW. LROSs, as before, are not decomposable. The structure of an LROS is identical to that of

an LASM, in that it is represented by a directed graph in which the graph nodes also correspond to LSs and graph edges correspond to LEs. An LRoS also has LC and LP sets, but it additionally has the *Import Data* (ID) and *Export Data* (ED) sets to specify data sent from and to the LS from and to its LRoS. This model is recursive because an LS is realized by an LRoS, which in turn is composed of a set of child LSs, each of which may in turn be realized by an LRoS. The recursion terminates when an LS is realized by means of an LUoW.

An example recursive LASM is given in Figure 6. In contrast to the example shown in Figure 5, we no longer need to specify the type of video compression used at the LASM layer (Figure 6a); instead, we ask for a “video I/O” service. Assume that in this example the system-wide resource management scheme finds two LRoSs for the video I/O service (Figure 6b), one where JPEG video compression is used, and another where MPEG compression is used. The system-wide resource management scheme can choose the one that is best at run time, based on the system state. For example, if the system-wide resource management scheme finds that computing resources are highly utilized but that communication resources are less utilized, the scheme may choose to use JPEG video compression. If the opposite is true, it may choose to use MPEG compression. In contrast to the previous example, the system-wide resource management scheme can now structure the application in different ways at run time, depending on the system state. As the system state changes, it may choose to realize the LS by means of the other LRoS. Note that a single recursive LASM eventually decomposes into a set of single-layer LASMs. In this case, assuming that all other LSs are realized by a single LRoS, there would be two single-layer LASMs, one using JPEG and the other using MPEG.

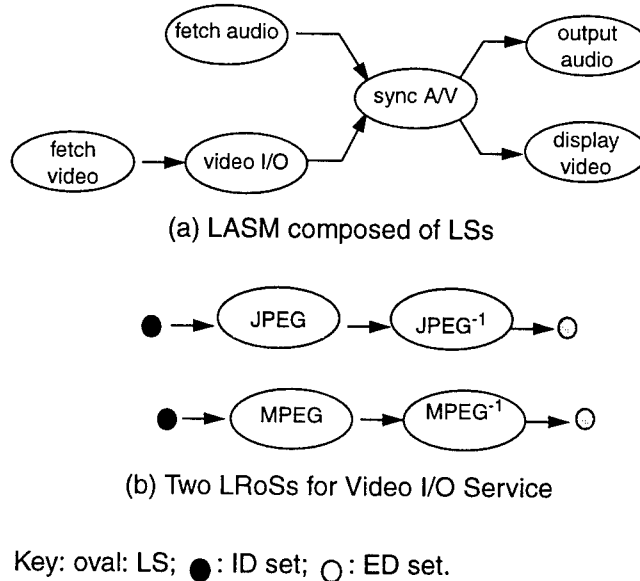


Figure 6. Recursive LASM

This recursive model enhances scalability. At the top level, the application developer specifies only that he or she needs to use a video I/O service. The specific details of this service can be specified by someone else. The model also enhances portability. If the application is executed on many different systems, each system can specify what video I/O services it can provide, without requiring any changes in the top-level application specifications.

3.2.2 System-Specific Application Stream Model

A user who wishes to execute an application invokes the application by (1) choosing the LASM corresponding to that application and (2) specifying QoS requirements for the application by assigning values to variables in the LP set via the AIM.

Once the user invokes the application, the system-wide resource management scheme must allocate resources to the application and schedule the application on those resources. The result will be the creation of a System-specific Application Stream Model. The SASM is specific to *a LASM, for a given invocation, on a specific system.*

Because not all system resources are interchangeable, not all resources can execute all LUoWs. Second, even those resources that can execute an LUoW will execute the LUoW in different ways and in different amounts of time, depending on resource-specific attributes. In this subsection we describe how the system-wide resource management scheme determines which system resources can execute each LUoW. Given this set of eligible resources for each LUoW, an allocation algorithm [Chatterjee 1997] can determine which resource to use on the basis of application QoS requirements and system objectives.

Each resource exports a set of Physical Units of Work (PUoWs), thereby identifying the LUoWs that the resource can execute. A PUoW is similar to an LUoW, except that it has the following additional attributes:

- **LUoW name:** the name of the LUoW it can execute
- **Resource demand model:** the load that executing this LUoW will place on this resource.

Note that this load specification includes more than just the amount of time necessary to execute the LUoW. Consider an LUoW executing on a CPU resource. Some of the load may execute in user-level space and some in kernel-level space; and some may be overhead, due to switching between the user and kernel levels. We use a complex task model to capture these load specifications [Colquist 1996; Harbor, Klein, and Lehoczky 1991]. This task model captures each of these incremental load components, its priority, its type (e.g., overhead or real work), and its memory requirements.

When the system manager attempts to bind a LUoW to a resource, it checks each resource's PUoW set. If any LUoW name matches the name attribute of the LUoW, then the LUoW can execute on this resource. Table 2 shows three example PUoW sets or three resources. The stub and line headings in the left-hand column show various LUoW names. The three right-hand column headings and subheadings at the top show various computing resources. The three table columns show the PUoWs provided by each resource. For example, we see that the JPEG LUoW can execute on either CPU 1 or CPU 2. Note that CPU 2 provides two implementations for JPEG, one using JPEG hardware and one using software. The complex task model for each PUoW, not shown, is tailored to each resource, based on the resource's attributes and the implementation.

Table 2. PUoW Sets

| LUoW Name | COMPUTING RESOURCES | | |
|----------------------|--------------------------|--|---|
| | CPU 1 SPARCstation 20 | CPU 2 SPARCstation 5 with JPEG Hardware | CPU 3 Pentium Processor with MPEG Hardware |
| JPEG | SW_JPEG | SW_JPEG HW_JPEG | |
| JPEG ⁻¹ * | SW_JPEG ⁻¹ | HW_JPEG ⁻¹ | |
| MPEG | | | HW_MPEG |
| MPEG ⁻¹ | SW_MPEG ⁻¹ | | HW_MPEG ⁻¹ |

*The superscript “-1” denotes a decompression algorithm.

These PUoWs can be viewed as templates. When the system manager determines which resource to be used for each LUoW, it creates an *instance* of a PUoW for each particular LUoW. This PUoW instance inherits the template attributes of its corresponding LUoW and PUoW. Unlike those of an LUoW or a PUoW in template form, all attributes of a PUoW instance are mandatory; optional attributes not assigned at the PUoW template or the LUoW level are assigned by the system-wide resource management scheme during allocation and scheduling.

A collection of PUoW instances defines the SASM. We also represent the structure of the SASM as a directed graph, where the PE is a set of edges indicating the order in which the PUoWs execute. Communication, implicit at the Logical Application Stream Model level, is now explicitly specified via PUoWs.

A set of PUoWs that provide a service captured by means of an LRoS is denoted a Physical Realization of Service (PROS).

An example SASM for an A/V teleconference is shown in Figure 7. The rectangular boxes correspond to PUoWs.

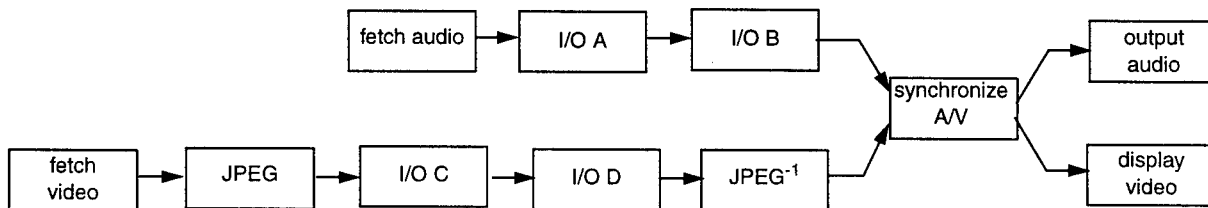


Figure 7. System-Specific Application Model Example

3.2.3 Benefit Functions

The user uses the AIM to specify QoS requirements for an application. Under certain circumstances, however, the system cannot provide this level of QoS. Rather than dropping the application entirely, the user may prefer degraded service. In this subsection we describe an abstraction, denoted the *Benefit Function*, that conveys application user QoS preferences to the system-wide resource management scheme, so that the system-wide resource management scheme can gracefully degrade application QoS so as to minimize the adverse impact to the user.

Our BF is a multidimensional function that indicates the benefit that the application user receives as a function of the QoS provided. The dimensions of the BF correspond to the variables in the LP set for the LASM. Figure 8 shows a simple BF for a movie-on-demand application. For simplicity, assume that this LASM has only two QoS parameters in the Logical Parameter set: frame jitter and frame size. The BF conveys the following information to the system-wide resource management scheme:

- The application user gains the most benefit if the frame jitter is small (less than 0.2) and the frame size is large (larger than 0.8).
- The benefit is constant in this top square in the plot.
- The benefit decreases linearly as the frame size decreases between 0.8 and 0.3 (normalized), if the frame jitter is held constant; the benefit is zero if the frame size is less than 0.3.
- The benefit decreases logarithmically as the frame jitter is increased beyond 0.2 (normalized), if the frame size is held constant.

All the dimensions of the BF have been normalized.

A conceptual description of how the system-wide resource management scheme may use a BF is given below. At any given time, the resource manager prunes the BF space to consider only points within the BF space (corresponding to QoS levels) that can be supported by the system, based on the system state and the load that the level of QoS places on the system (the load is determined from the PUoW resource demand model and the resource attributes). The system-wide resource management scheme then navigates this pruned BF space and finds the QoS that maximizes the benefit to the user.

For the example in Figure 8, multiple resource failures may not allow the system-wide resource management scheme to provide a frame size of 0.8 or a frame jitter of 0.2. In that case, the system-wide resource management scheme initially decreases the frame size because the user retains a greater benefit; once the frame size is reduced to 0.65, the system-wide resource management scheme then starts to increase the frame jitter; this alternation between reducing frame size and increasing frame jitter continues as more resources fail.

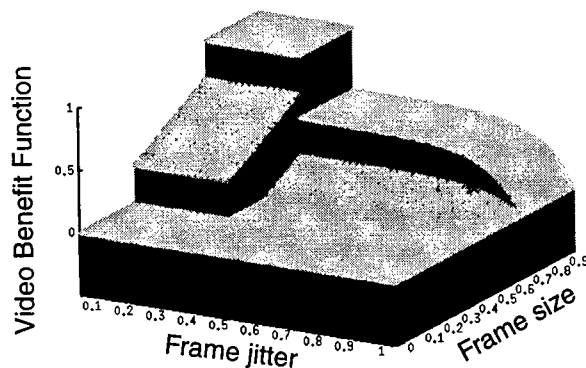


Figure 8. Video-Only QoS Benefit Function

Unlike the example BF in Figure 8, a real BF would have many more dimensions for QoS. Each dimension would correspond to a QoS metric of interest to the application user and would be changeable by the application. Therefore, each variable in the Logical Parameter set would correspond to a dimension of the BF. These dimensions would clarify the user's QoS preferences regarding flows; e.g., given a choice between reducing frame rate, picture quality, audio quality, and audio sampling rate, and increasing frame jitter and audio jitter, the user can specify the exact preferences.

Of course, each user would want the maximum QoS if the monetary cost were equal for each level of QoS. If the system is to garner the actual QoS that the user requires, each level of QoS must have an associated cost; therefore, the cost forms another dimension in our BF space.

In Appendix A and B, we present an alternative object-oriented model for capturing resource specifications. Future work may incorporate the best features of these two models into a single model.

3.3 RESOURCE MODEL

For the end-to-end support of the applications, it is important that the system be able to uniformly support the applications on multiple heterogeneous resources. The task of system-wide resource management is tractable only if the heterogeneous resources can be managed in a consistent manner. We model the resources for resource management by using an abstraction that hides the individual character of each resource and provides a uniform interface to all resources that is used by the management mechanisms. In this subsection we introduce this RM. The model has a resource-specific and a resource-independent set of attributes. The resource-specific attributes are internal to the resource and are used only by the internal system-wide resource management scheme. The external system manager does not have access to the internal details of the resource. The resource-independent attributes are exported on the resource interface, so that the management mechanisms can use the resource.

The RM describes the resources. It defines the resource types and the specific resources that are available to the system. The model also contains information about the resources that is required for the allocation algorithms, such as the speed of an individual processor or link. Each resource has a unique identifier and a classification type, as well as a set of parameters that define the performance and the security metrics of the resource. First, we formally define the resource, as follows.

Definition 1: A *resource* is a set of physical or logical devices (hardware and associated software) that can be controlled as a single unit by an internal **resource manager**. The resource also has an interface for external control. The resource provides a set of functions that can be performed on the devices.

Our definition of a resource encapsulates the physical device that is being controlled. Therefore, heterogeneous devices can still be defined to be the same resource with different parameters. Logical resources (nonphysical) such as stack or buffers used by the applications are also captured by this model. The resource-specific control of the resource is inside the resource and is performed by the resource manager. A resource-independent interface is provided for control from outside the resource. The external controlling agent resides in the system where the resource is located, and is independent of the specific resource attributes. This definition of the resource that

divides the resource management into a resource-specific management and a resource-independent generic management allows the incorporation of heterogeneous resources into the system in a uniform manner.

The system or an application identifies the resource, using a resource name or a type of service that is expected from a class of resources.

Definition 2: *A resource is uniquely identified by a **resource name**. The class of resource is defined by its **type**. The resource type identifies a class of resources that provide the same functionality. The type definition is hierarchical, and the properties of the parent type is inherited by the derived type.*

A specific instance of a resource is identified by its name. For example, a workstation may be identified by its *host name* (`beaver.erg.sri.com`), which acts as the resource name. The resource type of the workstation is `Sun Sparc Ultra-1`, a type of `Unix Workstation`, which is a type of `general purpose computer`. The resource type implicitly or explicitly identifies the class hierarchy in which the specific resources are instances. The properties of the class are inherited down the hierarchy.

As we remarked earlier, the resource has a set of internal attributes that are hidden from the external world and a set of exportable attributes that are made available for the external control and management mechanisms. The first set of internal resource attributes are related to the performance of the resource. The performance measure is very specific to the resource type; therefore, the resource vendor is hidden inside the RM.

Definition 3: *The **resource parameters** define the relevant performance metrics and the security levels for the devices in the resource. The resource parameters are a set (p_1, p_2, \dots, p_n) in which each p_i is the specific resource parameter. The parameter may be specified as a scalar, or as a random variable with a distribution function, or as a scalar-valued function.*

Associated with every resource is a set of performance metrics of the component devices. The metrics may be specified as a range (between the minimum and the maximum, represented by two parameters in the set). For example, the disk drive can be characterized by the minimum and the maximum seek time. The parameter can also be defined as a random variable with a distribution; e.g., the latency in a communication link can be described as a random variable with a mean value and a variance. In some cases it may be necessary to define a parameter as a scalar-valued function. The interpretation of the resource parameters depends on the specific resource profile; e.g., the profile will specify that parameter p_k must be interpreted as the maximum latency. All resources of a given type have the same profile. In addition to the performance metrics, each device in the resource has a range of security levels.

The other internal resource attribute of importance is the resource scheduling model. The resources have their own internal scheduling constraints, which affect the QoS behavior of the resource. The concurrency behavior of the resource is of importance for QoS-based resource management. The behavior of the resource when multiple applications are trying to use the resource directly affects the QoS guarantees the resource can provide to the different applications that use the resource. These behaviors are captured in the scheduling model. For example, consider a resource comprising an operating system running on a desktop machine. The scheduling model captures the behavior of the operating system, the CPU, the internal buses, and the disk drive, as well as how this behavior affects the QoS provided to applications running on this resource.

Definition 4: *The resource scheduling model describes the scheduling behavior of a resource and how it affects the QoS requirements of applications that use the resource.*

The exportable resource attributes abstract the internal workings of the resource and present a standard interface for the system control and management infrastructure. The generic resource attributes, as defined below, fulfill the requirement of abstracting the specific details of the different resources and providing a common interface for handling heterogeneous resources.

Definition 5: *The generic resource attributes describe the performance of the resource with respect to the management of QoS in a resource-independent manner.*

The definition of the generic resource attributes provides only a “container” for the interface definitions of many different attributes. The specific attributes referred to by the generic resource attribute are allowed to be flexible and expandable at a later date. Currently, we have defined a subset of the generic attributes as resource capacity, QoS load, resource utilization, and resource reliability. The formal definitions of each attribute follow.

Definition 6: *The resource utilization measures the percentage of time that the resource is non-idle. The resource utilization is measured over a given time interval of interest.*

The resource utilization attribute measures how well the resource is being used. This value depends on the scheduling principle of the resource manager and the profile of the applications running on the resource. For example, suppose that two tasks are running on a single resource: Task 1 runs for 30 s every 100 s, and Task 2 runs for 100 s every 1,000 s; then the resource utilization is $(30/100 + 100/1,000) = 0.40$. In other words, the resource is idle 60% of the time.

Definition 7: *The resource capacity quantifies the total capacity of a resource at a parameter setting (specific values of the parameters). The resource capacity is specified in a resource independent set of units. The resource capacity is a function of the resource parameter set p_1, p_2, \dots, p_n .*

The resources can be operated at different parameter settings; associated with each of these settings is the capacity of the resource. The resource capacity measures how one parameter setting compares with the other settings. For example, a raid disk can be used as a simple storage device with no replication and a large capacity; or it can be operated as a storage device with inherent fault tolerance because of replication across parallel disks, but with a lower storage capacity. The capacity has to be specified in resource-independent units; e.g., the computational capacity can be specified as the MIPS* or FLOPS rating of a standard processor. The resource manager can compare the total capacity of the resource to the current usage and determine if a new piece of work can be assigned to the resource.

Definition 8: *The QoS load attribute measures how well a resource will meet an application's QoS requirements. The measure computes a tolerance between the current QoS requirement of the application and the ability of the resource (in its current state) to provide the QoS before the resource misses the QoS requirements.*

The QoS load attribute is a generalization of the concept of schedulable saturation, in that it measures how much “slack” exists between the current ability of the resource to provide the QoS requested by the application, and when the resource starts to miss the QoS commitments, or how

*MIPS: million instructions per second.

much room the resource has to maneuver the application tasks before it starts to miss its QoS commitments. The QoS load attribute is a function of the resource parameters, the resource scheduling model, and the characteristics of the current tasks on the resource.

Definition 9: *The **resource reliability** attribute measures the probability that the resource will provide the correct results for the requested service.*

The resource reliability attribute tries to capture the assurance the system can get from a resource about the availability and the validity of the results. The reliability measure aggregates the effects of the different anomaly-causing phenomena, which may range from the resource becoming inaccessible (because a fault causes it to go off line), to the introduction of errors into the results because of certain environmental effects.

Definition 10: *The resource model is a tuple $[N, T, R, P, S, A]$, where N is the resource name, T is the resource type, R is the resource manager, P is the resource parameter set, S is the scheduling model, and A is the generic resource attributes. In addition, the resource model has the functional and control interfaces for the resource.*

The RM captures all of the above dimensions of the resource into a formal model for use by the system manager. The RM is defined to include the resource name, the resource type, the resource manager, the resource parameter set, the resource scheduling model, and the generic resource attributes. The RM also has the functional and control interfaces for the resource.

3.4 SYSTEM MODEL

From the system's point of view, multiple LASMs must be supported on the resources available in the system. A distributed control and management structure is needed to manage the resources globally. The LASMs decompose into the constituent LUoWs and services; these individual LUoWs and LRoSs must be scheduled on the different resources. The motivation for creating the SM is to define a structure of the system in an implementation-independent manner that

- Supports distributed applications with different end-to-end QoS requirements
- Allows the implementation of a scalable and tailorable system manager
- Encompasses the management of heterogeneous resources within the same system structure
- Defines independent domains of management for the local control of the resources and applications.

The key to the SM is the hierarchical decomposition of the system to provide "encapsulated domains" of management. The system is composed of subsystems that are themselves composed of subsystems. At the bottom of the hierarchy, the subsystems encapsulate a set of resources that are defined by their RM. This hierarchical description of the system allows the resources and the applications to be managed at different levels of granularity.

Definition 11: *A system is uniquely identified by its **system name**. The system also has a **type** that defines the class of the system. The system type is the union of the type classifications of the component subsystems.*

As defined above, the resources have a type that identifies the class of the resource. The subsystems in the hierarchy that contain the resources inherit the type definitions of the resources. Therefore, the *type* of the subsystem is the union of the type definitions of all its resources. Then, by recursion, as we move up the system–subsystem hierarchy, the subsystems inherit the type definitions of their component subsystems.

Definition 12: *A subsystem is uniquely identified by its **subsystem name**. The subsystem also has a **type** that defines the class of the subsystem. The subsystem type is the union of the type classifications of the component subsystems or of the component resources.*

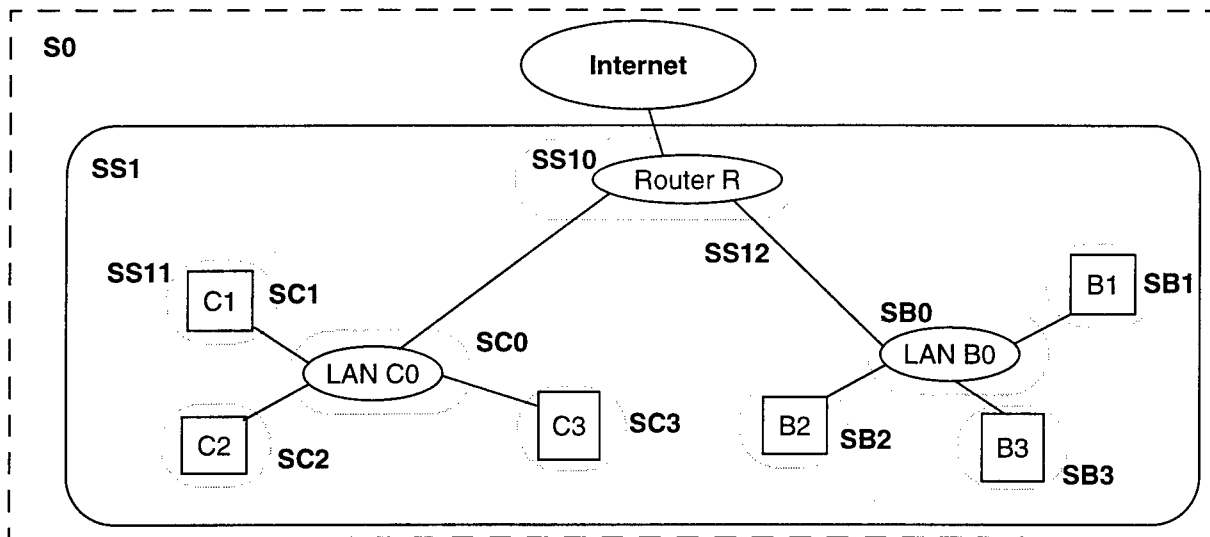
Definition 13: *A system S is a tuple $[N, T, \Sigma, E, M]$ where N is the unique name of the system, T is the type of the system, Σ is a set of subsystems, and E defines the connectivity between the subsystems. The system is represented as a graph; the elements of Σ are the nodes and the elements of E are the edges. The last component of the tuple, M , is the system manager that manages the subsystems Σ .*

In the definition hierarchy for describing the distributed system of interest, the *system*, S , lies at the top of the hierarchy. The system is represented by a graph that defines the component subsystems and the topology. The system manager, M , is a logical entity that manages the system with regards to allocation, and scheduling of application tasks in the system. The system manager is responsible for the QoS assurances given to the applications. The basic unit of management and system definition is the *subsystem*.

Definition 14: *A subsystem is a tuple $[N, T, S, \Sigma | \Phi, E, A, m]$ where N and T are the name and the type of the subsystem. S is the parent (sub)system of the subsystem, Σ is a set of subsystems, and m is the subsystem manager. Φ is a set of resources. The subsystem either has a set of subsystems Σ or a set of resources Φ . The subsystem manager therefore manages either the constituent subsystems or the resources. E is the set of edges that define the connectivity between the subsystems or the resources (as the case may be). A is the set of generic subsystem attributes.*

The system is composed of multiple *subsystems*. The subsystems are defined recursively to contain subsystems or *resources*. A subsystem either contains all subsystems or has all resources. A subsystem may have one or many resources; all of the resources within a subsystem are managed together by the subsystem manager. The control and management of the child subsystems is different from the control and management of resources; therefore, we do not allow a single subsystem to have both subsystems and resources. Logically, this restriction should not pose a problem in modeling any arbitrary system, since we can always model a resource as a subsystem with has only one resource.

The hierarchical structure helps in the control and management of the global resources in the entire system. At the bottom of the hierarchy are the resources in the system. A subsystem is represented by a graph in which the nodes represent the child subsystems and the resources, and the edges are the connectivity between them. Two subsystems, S_1 and S_2 , are connected if a physical connection exists between them. A physical connection between subsystems implies that there exist resources in the descendent hierarchy of S_1 that are physically connected to resources in the descendent hierarchy of S_2 . Figure 9 shows an example system with its component subsystems and their resources. Figure 10 shows the hierarchical graph structure representing the example system, derived from the resource and system topology.



The global system at the top of the hierarchy is S0. SS1 is a subsystem consisting of two LANs (SS11 and SS12) and a network router (encapsulated in Subsystem SS10). The subsystem SS11 has four subsystems (SC0, SC1, SC2, and SC3), each of which has a single resource (C0, C1, C2, and C3, respectively). Similarly, SS12 has four subsystems (SB0, SB1, SB2, and SB3), with their respective resources B0, B1, B2, and B3).

Figure 9. Example System with Its Component Subsystems

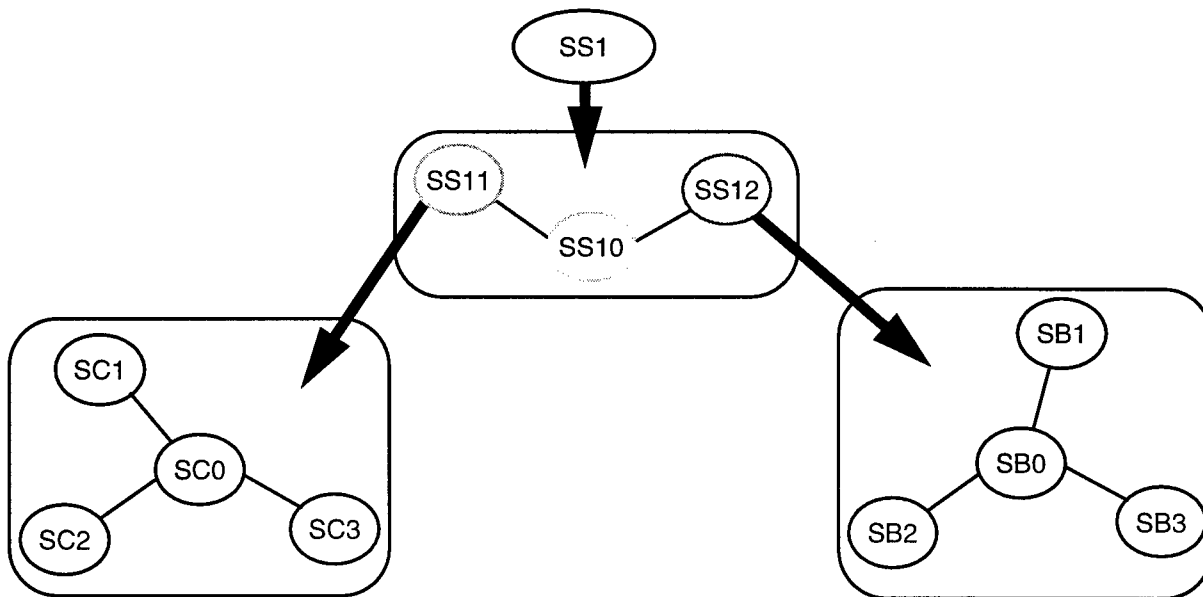


Figure 10. Hierarchical Graph Structure Representing the Example System, Derived from Resource and System Topology

The generic subsystem attributes are aggregations of the generic resource attributes of the component resources or the aggregations of the generic subsystem attributes of the component subsystems. The component attributes are defined in the same way as the resource attributes.

3.4.1 Subsystem Interfaces

Subsystems manage the resources or the component (child) subsystems. The subsystems interact with each other and the parents through the interfaces. The resources internal to a subsystem are available to the parent system or subsystem through functional and management interfaces. Each subsystem or resource has four types of interfaces (as shown in Figure 11): functional, QoS and negotiation, management, and instantiation.*

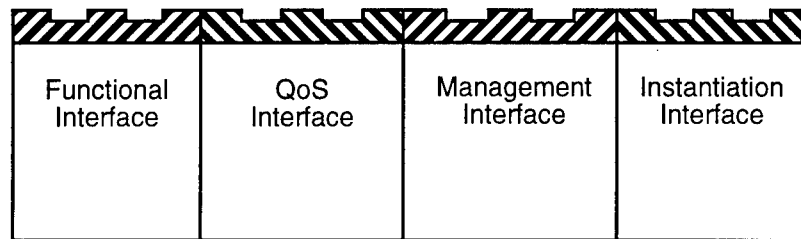


Figure 11. Subsystem or Resource Interfaces

3.4.1.1 Functional Interfaces

Functional interfaces define the operations (or services) that a subsystem or resource performs for the system or for applications. These interfaces represent the functionality supported by the subsystem. A subsystem has several functional interfaces. The functional interfaces are defined by the applications and the system services.

The functional interface of a subsystem is derived from the functional interfaces of the system's component subsystems and resources. This subsystem can support all the functions of its component subsystems and their resources. In addition, the functional interface of the subsystem also supports new functions derived from the composition of the component subsystem functions. The PROs existing at the subsystem are accessible, through the functional interface. In addition, an LROs can be submitted to the subsystem in a generic manner, through the functional interface.

3.4.1.2 QoS and Negotiation Interfaces

QoS and negotiation interfaces are service specific interfaces for QoS specification and negotiation, as follows. An LROs (or an application) presents a set of subsystems with a request for a service (e.g., a procedure call) at the functional interface. The associated QoS parameters are presented at a QoS and negotiation interfaces; the latter interface also handles the negotiation between the application and the subsystem. The QoS and negotiation interfaces deal with dynamic information about QoS. The QoS and negotiation interface aggregates the dynamic information about achievable QoS for each service on a periodic basis (or when the values change).

3.4.1.3 Management Interfaces

Management interfaces are used for communication with subsystem managers. The same interfaces are also used for the installation of new policies, if the allocation routines are policy driven. Monitoring for the failure of resources and the dissemination of information about such failures are another responsibility of the management interface.

*These definitions are similar to those of the Object Management Group, Inc. (OMG) CORBA service interfaces.

The primary function of the management interface is its role in the control of subsystems: it is the interface through which parent subsystems control their child subsystems and the associated resources. The control functions include the following:

- Starting, stopping, and aborting the execution of a PRoS
- Scheduling and allocation of PRoSs and the PUoWs
- Reallocation and rescheduling when the system state or resource state changes
- Invocation of the negotiation and renegotiation process
- Resource reservation.

3.4.1.4 Instantiation Interfaces

Instantiation interfaces enable new services to be implemented in a subsystem. For example, there may be a need to extend the services being provided by a subsystem: accordingly, a new physical realization of the service is instantiated in the subsystem. The PRoS actually is implemented as PUoWs on the resources at the bottom of the subsystem hierarchy; an application LUoW is mapped to each of the PUoWs, which together form PRoSs that correspond to the LRoS of the application. These PRoSs are instantiated at the subsystems' instantiation interface, through which the segment of code that implements a new PUoW also is installed on a resource through this interface.

3.5 CORRESPONDENCE BETWEEN APPLICATION AND SYSTEM MODELS

There are close parallels between the models that describe the applications and the systems. The recursive application model shows the global structure of an application at the top level; as we go down the levels, the specific details of the application's components become visible. Similarly, the SM at the higher levels shows only a global structure of the subsystems while hiding the details about the local resource topology. As we move down the hierarchy, the detailed structure becomes more apparent.

Applications and the component LRoSs are associated with specific subsystems and the resources. We associate an LRoS with a resource or a subsystem if a PRoS implementing the LRoS is scheduled on the resource or subsystem, and if the resource is allocated to the PRoS. The services instantiating the LRoS are associated with the child subsystems of the subsystem. This association continues recursively until the LUoWs are implemented as a PUoW on a specific resource at the bottom of the subsystem hierarchy. The allocation and scheduling algorithms schedule the PUoW on the resource. The allocation and scheduling binding at the lowest layers of the application and of the SMs is propagated up the models to create associations between the LRoS at any level and subsystems at any level.

The primary function of the system-wide resource management is to associate the LRoSs and the LUoWs to the specific physical resources. The PRoSs and PUoWs corresponding to the LRoSs and the LUoWs are allocated the resources and scheduled for execution. The resources are then monitored for proactive and reactive management in the presence of faults. The system management policies determine which management mechanisms must be invoked in different situations.

A subsystem provides a set of services by using the child subsystems or the resources under the management domain. The services are available to the applications through a set of interfaces of the subsystem. If the subsystem manages resources, then the interfaces are based on the functional interfaces of the resources. In the case of subsystems managing child subsystems, the interface definitions are derived from the interfaces of the child subsystems. The available services in a subsystem are registered with the subsystem manager in a manner similar to CORBA brokers.

The association of an LRoS with a subsystem or a resource is based on the functional description of the LRoS, the QoS needs of the LRoS, and the services supported by the resources. Each subsystem provides some PRoSs through the functional interface. These PRoSs can be used by the applications or the parent subsystems as the PRoS needed in the LRoS. The subsystem interface hides all the implementation details of the realized service from the rest of the system and the application.

Figure 12 shows the mapping of a simple application to a system composed of three subsystems and two resources encapsulated inside subsystems. The services encapsulating the LUoWs U1 and U2 are mapped to the subsystems encapsulating the resources R-1 and R-2 via their implementation as PUoWs executing on the resources R-1 and R-2. The services S1 and S2 are mapped to the subsystem SS-3, and the service S3 is mapped to the subsystem SS-1. The mapping is recursive, i.e., the components of the services (S1, S2, and S3) are mapped to the components of the respective subsystems. Suppose that S1 is a compression service, and SS-3 has a compression service as one of its components. If the service (say S1 is a compression service) is equivalent to a PRoS that is provided by the subsystem, then the mapping is completed at that level; otherwise, the mapping continues all the way down both the hierarchies until the PUoWs are mapped to the resources.

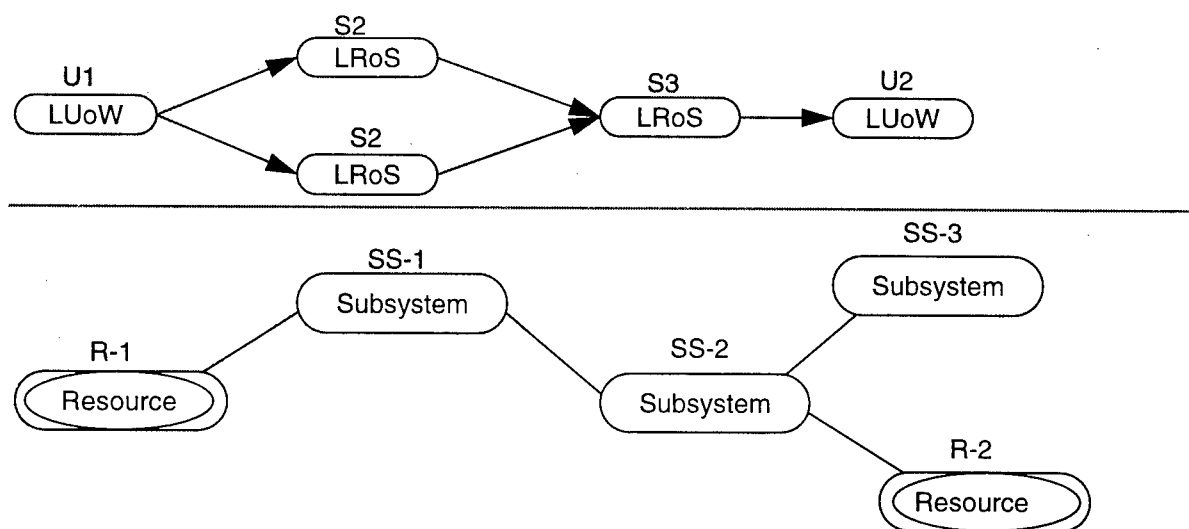


Figure 12. Mapping of An Application (LRoS) to Resources and Subsystems

3.6 DISCUSSION

In this section we have presented a model for describing distributed systems that supports distributed applications with QoS guarantees. The hierarchical SM allows different types of systems with distributed, centralized, or hybrid control schemes to be modeled by means of a common set of constructs. The primary purpose of the model is to enable resource management to be performed in an intelligent manner that compensates for changes in the system state and guarantees QoS to the applications being executed on the system.

The hierarchical structure of the SM complements our recursive application model. Information about the end-to-end application state and the state of the resources being used by the application can be considered the “first order” of information, since it directly affects the performance of the application, from a user’s perspective. First-order information plays a critical role in adaptive resource management. In addition, other, “second order” information, such as the state of other applications in the system and the state of other resources that may affect the current application, is needed by the managers. Second-order information affects the performance of the applications indirectly: for example, resource fault that is not related to a given application, may spill some of the load from that resource to the resource being used by the application, and thus may degrade the application QoS. The subsystem managers at the different levels of the subsystem hierarchy consider the global and local state of the applications and the resources at different resolutions.

Another feature of the SM and RM is that they hide many implementation-specific details behind interfaces, which enables heterogeneous resources to be modeled within the same SM. Resource-specific details are hidden inside the RM, and the system uses resource-independent interfaces to interact with the resources. This approach also enables uniform management mechanisms to be used for heterogeneous systems.

4 ADAPTIVE RESOURCE MANAGEMENT ALGORITHMS

Our resource management architecture, previously described in Section 1, is reshown below in Figure 13. In Section 3 we have described the application, invocation, resource, and system model (AIM, RM, and SM) components of this architecture. In this section we describe the five algorithm components of the architecture. The algorithms and the models together form a resource management scheme and provide admissions control and negotiation.

The applications and users interface with the rest of the system via the *end system*, a subsystem on which a user can invoke an application (run a QoS-aware application). The goal of comprehensive end-to-end resource management is to be able to admit new applications, reserve resources for the application, allocate resources for the tasks, and schedule them on the system, and to do so in a manner that satisfies the QoS needs of all the applications in the system. Further, the system must deal with many unpredictable events such as random application arrival times, resource failures, and security compromises.

Adaptive resource management requires information about the states of the different resources in the system and the states of all the applications being serviced. Since the user is interested in the end-to-end performance of the applications, the internal states of the individual PRoSs must be aggregated to provide an estimate of the global application state. Tolerances in the internal states of the individual PRoSs and the PUoWs can be allowed, as long as the end-to-end QoS guarantees are fulfilled. This implies that the system manager needs the global state of the application to evaluate the performance of the system with respect to a given application. However,

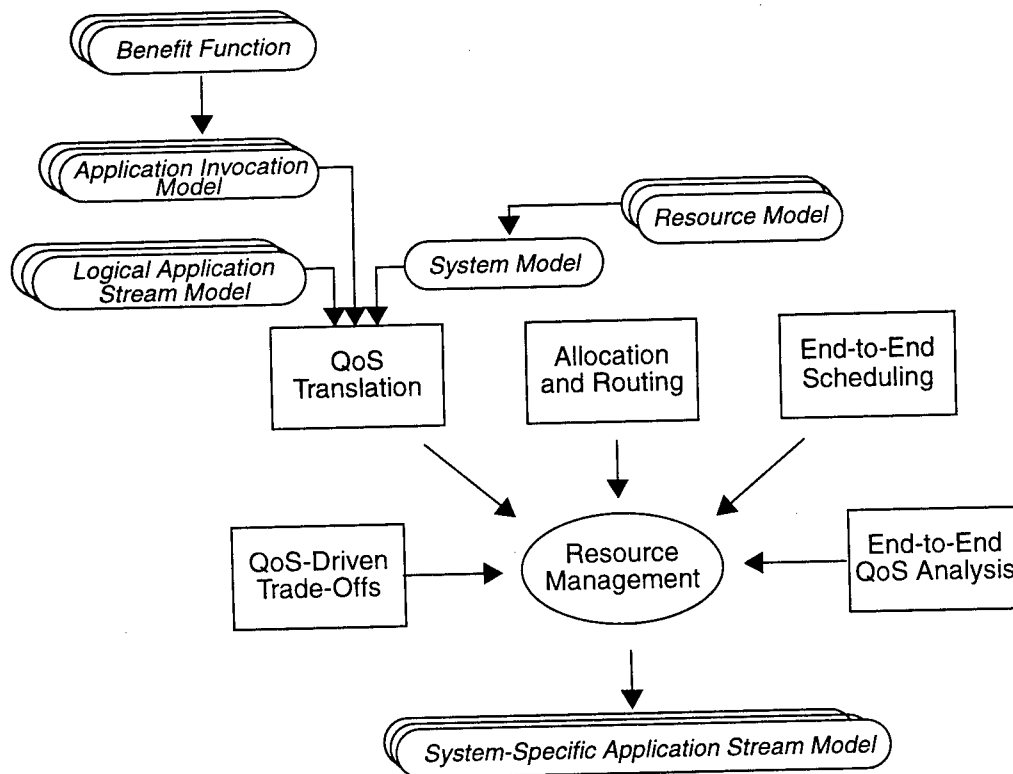


Figure 13. Resource Management Architecture

in the management of the specific resources, it is more important to obtain the specific (local) resource state information than to obtain aggregated information about the collective resources available. The aggregated global state may indicate that sufficient resources are available, but a specific application may still fail if the resource allocated for the application's PUoW fails during execution. In this section we describe how the hierarchical system model helps in resource management by allowing information about the resource states and the application states to be available at different resolutions. As we move up the hierarchy in the hierarchical system model, global information on the states of applications and resources; as we move down the hierarchy, local information about the resources and the application PRoSs and PUoWs is available.

The orchestration of multiple applications in a dynamically changing system requires careful coordination among the subsystems. Information about the state of the subsystems and the applications must be maintained by the system. This information is generated in a distributed manner. Internally, each subsystem has information only about its own resources and PUoWs. The internal information is distributed across the system when the parent subsystems exchange information with their child subsystems. The hierarchical structure allows the distributed management to consider both local system and applications state information and the global state of all the applications and the subsystems, at different resolutions and different levels of aggregation. The subsystem managers at higher levels of the hierarchy have global state information collected over a larger span than that of the local state information available to subsystem managers at the lower level of the hierarchy.

An event in the system such as the invocation of an application at the end system, or a fault in a resource, or the completion of an application, triggers management algorithms in the system. When a new application is invoked at an end system, the subsystem manager negotiates with the application before admitting the application into the system. The *admission process* compares the resource needs and the service requirements, including QoS, of the new application with the available resources and services on the system. The available resource capacity is the difference between the total capacity of the resource and the aggregate resource needs of multiple applications using the resource:

$$R_{available} = R_{total} - \sum_{a \in A} R_{usage}(a)$$

where the set A is the set of all applications using the resource R . It should be noted that the aggregation process for resource usage may not be a simple summation process. The admission process also considers the different constraints the application may have (e.g., a need for special hardware) before making the admission decision. Information about the resource capacity at the higher levels of the subsystem hierarchy is obtained from the lower levels of the hierarchy. At each level the subsystem aggregates the information from all the child subsystems and passes this to the higher levels. Gross management decisions are taken at the higher levels of the subsystem hierarchy; these decisions are subsequently refined down the lower levels. Only at the resource level can the system make a final commitment. The management decisions are taken by the subsystem managers and the resulting information is propagated up and down the hierarchy.

A reservation mechanism can be used by the system to ensure that the application needs are always satisfied. The *reservation process* estimates the resource needs of the application and, using a *reservation protocol*, guarantees the resources to the applications. The specific resources that an application uses are identified by an *allocation process*. The allocation mechanism maps the QoS needs of the application and its constituent LRoS to specific resource needs. Constraints

imposed by the applications are also considered in the allocation of specific resources to the application. The flow of the application is analyzed in order to schedule the specific tasks on the resources at different times. The *scheduling process* uses the resource demands of the application services and the resource capabilities of the allocated resources to create a schedule. Again, since information about specific resources is hidden inside the aggregate information at the higher-level subsystems, the final resource commitment is made by the allocation process only at the level of resources.

If the application needs cannot be satisfied by the current state of the system, or if, during the execution of the application, an anomaly occurs, then the application tasks must be *rescheduled*. The system can invoke a *negotiation process* that understands the needs of the application and the capabilities of the system, in order to secure an attainable service level for the application. The *benefit function* is used in the negotiation process to ensure that the application obtains a satisfactory QoS.

The subsystem manager interacts with the child subsystems to obtain the status of the subsystems and the available services. This information is available through the different subsystem interfaces. The subsystem manager and the global resource management process decide the extent of the distribution needed to support an application. A larger set of resources is available to support an application for subsystems at the upper levels of the hierarchy; however, the number of contending applications is also larger. The tradeoff is between the work needed to search a larger subsystem to find a better allocation and schedule, and the benefit obtained by the better solution. It is evident that a global optimal resource allocation and schedule cannot be attained; therefore, methods that provide satisfactory solutions within certain time and space boundaries should be explored.

In the subsections below, we describe (1) the objective of each algorithm, (2) previous work in this area, (3) missing components necessary for end-to-end adaptive resource management, and (4) our approach to the mapping of application QoS requirements to resources.

4.1 QOS TRANSLATION

The QoS translation algorithm translates the end-to-end application QoS requirements presented to the top-level resource management scheme to QoS requirements for lower-level resource management schemes. Eventually, the translation reaches the bottom layer, where the translation algorithm specifies the QoS requirements for a set of resources. This information is used by subsystem-level and resource-level resource management schemes to provide the necessary QoS support.

QoS translation is based on QoS taxonomy. Our QoS taxonomy, described in Section 2, identifies the interrelationships between various QoS metrics and specifies how a QoS metric at one level affects QoS metrics at other levels. For example, loss of precision at the network layer translates into loss of accuracy at a higher layer. This information is used by the QoS translation algorithm to make intelligent decisions.

A general QoS translational scheme for timeliness- and precision-related QoS metrics was described by Chatterjee and Strosnider [1995c]. A similar approach can be utilized for incorporating other QoS metrics, including accuracy, availability, and security, into a comprehensive translation algorithm. An example translation scheme for a teleseminar application is provided in Subsection 4.7.

4.2 INTEGRATED RESOURCE ALLOCATION AND ROUTING

The objective is to develop an algorithm that will allocate computing, storage, and communication resources to applications, in order to meet application QoS requirements and to optimize a system objective function.

Work has been done in both QoS-based routing and QoS-based allocation. Routing algorithms attempt to find a communication route between a fixed source and destination that either minimize end-to-end latency [Parris and Ferrari 1993] or minimize the number of I/O hops [Rampal, Reeves, and Agrawal 1994]. It is our view that this approach leads to inefficient resource usage. Many applications have QoS requirements—although failing to satisfy the QoS requirements is deleterious to the application user, far exceeding the QoS requirements does not elicit a greater benefit to the user than barely satisfying those requirements. On the other hand, systems attempting to optimize application QoS frequently utilize resources that could have been utilized by other applications. Therefore, our view is that resource management algorithms should be based on a different philosophy—instead of finding a path that *optimizes* application QoS, these algorithms should find a path that *meets* application QoS requirements and *optimizes* the system objective (e.g., balances the system load or minimizes system utilization).

The QoS-based allocation algorithms attempt to find a set of computing resources to allocate to each application. Most of these algorithms [Nanda and Stenger 1991; Ramamrithan 1989; Tindell, Burns, and Wellings 1992] assume that the system is composed entirely of homogeneous processors connected by a local-area network (LAN), simplifying the problem greatly. Some of these algorithms [Nanda and Stenger 1991; Tindell, Burns, and Wellings 1992] are also based on simulated annealing and cannot be used for on-line allocation. Others [Ramamrithan 1989] do not consider end-to-end latency constraints.

These algorithms therefore cannot be used for on-line resource management of heterogeneous resources connected by an arbitrary communication topology. The arbitrary communication topology assumption requires the allocation algorithm to simultaneously identify communication paths for data transfer between these computing resources. In general, an integrated allocation and routing algorithm, one that simultaneously identifies the set of heterogeneous computing and communication resources to utilize for each application, is required. Such an algorithm was developed by Chatterjee and Strosnider [1997]. The algorithm was based on a hierarchical application of Dijkstra's shortest-path algorithm [ibid.], with the key problem being the assignment of resource weights on the basis of application QoS, system objectives, and resource-level and end-to-end scheduling policies. This algorithm addressed only hard real time applications, considered only timeliness and precision metrics for application QoS, and did not adapt to changes in the system state caused by resource failures or application arrivals or departures.

The above algorithm should be improved in a two-step process. In the first stage, the problem should be constrained to the allocation of resources to a single application, based on its comprehensive QoS requirements and on the system objective function, but without support for migrating existing applications. In the second stage, the algorithm should support the migration of existing applications, in order to derive a better (near-optimal) solution. We can then compare the optimality of the two solutions and consider the results in light of the extra run time of the second solution, to see whether migration is worthwhile. In the remainder of this section we discuss our approach to these two stages of this process.

Initially, one should develop a resource weight assignment methodology that can be applied to hard, soft, and non-real time applications, with temporal, accuracy, and precision QoS metrics. The new resource weight assignment methodology should be applicable to a variety of scheduling policies and to systems comprising heterogeneous, real-time operating systems and real-time network protocols.

During the initial approach, one should tackle the migration problem. Key to the solution of this problem is to determine which existing applications to reallocate and reroute so as to achieve better results. At the same time, it is imperative that we consider the run time of the algorithm, because the algorithm must execute on line.

Several approaches to this problem can be examined. One might be to use simple case-reasoning algorithms to determine which applications to reallocate and reroute. For example, if an application must use a certain resource, then all other applications on that resource can be reallocated and rerouted. Another potential approach is to apply microeconomic theory to this problem. Microeconomic theory models a free market containing consumers (applications) and producers (resources). Allocation is determined by the metaphor of a market, where competing applications negotiate for bundles of resources that can be converted into services at differing levels of QoS. There are several advantages to this approach, including the ability of this free market to allocate resources in a decentralized manner, so as to minimize the information that needs to be conveyed between resources.

4.3 END-TO-END SCHEDULING

An end-to-end scheduling and flow control policy that works uniformly over all computing, storage, and communication resources is required for predictable behavior in a distributed system.

Application execution over a set of computing, storage, and communication resources can be modeled (1) as a *distributed pipeline*, and (2) to specify non-work-conserving scheduling policies* between distributed pipeline stage boundaries [Chatterjee and Strosnider 1995c]. These measures will make the system behave in a predictable manner. Because previous work in this area has considered only hard real time applications, and timeliness' and precision-related QoS requirements [ibid.], a need exists for a more complex, distributed pipeline scheduling algorithm that also considers accuracy and availability QoS metrics and provides integrated support for diverse applications. Aperiodic servers must also be incorporated within the scheduling mechanism, to provide QoS support for distributed transactions.

*Non-work-conserving scheduling may leave a resource idle even if work is pending on that resource.

4.4 END-TO-END QOS ANALYSIS

QoS analysis is needed to determine whether application QoS requirements will be met by the system. We have developed a hierarchical QoS analysis algorithm [Chatterjee and Strosnider 1995a] that decomposes the complex problem of end-to-end QoS analysis into a set of simpler, independent QoS analysis problems. These independent problems can be solved in parallel, leading to a scalable and efficient solution. In addition, the hierarchical character of this algorithm enables us to easily analyze a large-scale system composed of heterogeneous computing, storage, and communication system resources.

The current algorithm [Chatterjee and Strosnider 1995a] supports only deterministic timing analysis for hard real time applications. The algorithm needs to be augmented to support statistical and deterministic analysis of hard, soft, and non-real time applications; accuracy, and availability QoS metrics must also be considered.

4.5 QOS-DRIVEN TRADE-OFFS

QoS-driven trade-offs will enable the resource management scheme to adapt to changes in system state. For example, an application may specify that it can support both MPEG and JPEG video compression. In that case, the system-wide resource management scheme should decide, on line, which compression technique to use, based on the relative loads of computing and communication resources. If at some time t the communication resources are the bottleneck but computing resources are plentiful, then MPEG would be a better choice; if computing resources later become the bottleneck and communication resources become plentiful, JPEG may be better. If both types of resources are scarce, then the algorithm must degrade the QoS provided to the application.

QoS-driven negotiation and graceful degradation are based on intelligent trade-offs between various QoS metrics, to maximize the benefit received by the application user. Making intelligent QoS trade-offs requires an understanding of user QoS preferences and the corresponding resource usage requirements. Our multidimensional benefit function abstraction [Chatterjee et al. 1997] can be used to capture these QoS preferences. Conceptually, making QoS trade-offs is a matter of traversing this benefit function QoS space to find an operational point (a set of QoS levels) that maximizes the benefit to the application user and can be supported by the system. Since the representation and traversal of arbitrary multidimensional functions is prohibitively complex, representations that approximate the benefit function, and an algorithm to search this simpler BF space, must be developed. One possibility is to decompose an n -dimensional BF space into a set of 3- or 4-dimensional spaces, if the QoS metrics have low statistical correlation.

This algorithm should be developed in two stages. In the initial stage, the problem should be constrained so that only the applications whose QoS cannot be supported are degraded. The key focus in this stage should be on the development of an algorithm that will efficiently traverse the multidimensional BF QoS space. In the second stage, the application will consider degrading other applications to provide a solution that is closer to the global optimum. Microeconomic theory may be applied during the second stage of QoS-driven trade-offs. In the market metaphor, the BF expresses the preferences of each application, analogous to a consumer's utility. Each application's priority is expressed by its initial endowment, which, as a consumer, it can spend on acquiring resources. Quantities of resources in turn produce a level of service, as measured by QoS metrics.

The resource allocations reached at market equilibrium impute a price to each resource. This is a direct application of market equilibrium in a production and consumption economy [Varian 1978].

4.6 EXAMPLE

This subsection shows how the system-wide resource management scheme, which comprises the algorithms described above, will use the BF, the LASM, and the AIM in conjunction to instantiate an application on system resources. The LASM is described in Subsection 4.6.1. The PUoW Database is described in Subsection 4.6.2. The admissions procedure and allocation is described in a conceptual manner in Subsections 4.6.3 and 4.6.4.

4.6.1 Logical Application Stream Model

A LASM for a video teleconference application is given in Figure 14(a). The application samples audio at the source, filters out background noise, synchronizes the audio with the sampled video, and presents the audio and video at the destination. The video teleconference application is composed of eight LRoSs. For this example, we will pay particular attention to two of the LRoSs, the Video I/O and the filter audio services.

Figure 14(b) shows two LRoSs that realize the video I/O service; Figure 14(c) shows an LRoS that realizes an audio filter service. For the purposes of this example, we assume that each of the rest of the services is realized by a single LUoW. (These LUoWs are not shown in the figure.) In reality, each of these logical services would be realized by multiple LRoSs; each child service within each of these LRoSs would be realized by other LRoSs.

The filter audio LRoS shown in Figure 14(c) consists of two independent data flows. The first data flow receives data from the parent service, and the second, for garbage collection, executes independently. The two, however, are linked by the semaphores $P(S1)$ and $V(S1)$.

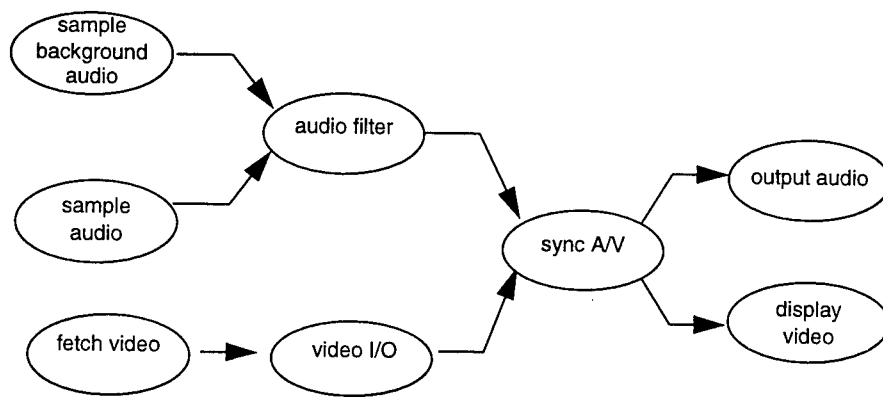
4.6.2 System Specifications

Assume that we use the system given in Figure 15. Entries in the PUoW database for selected resources are given in Table 3. Each row in the table represents an LUoW from the LASM. Each column represents a specific resource from the SM. Each entry (occupied cell) in the table represents the PUoW that corresponds to that LUoW on that resource. The identifier within the cells are used in this following example. Some cells are empty because the LUoW for that row does not have a corresponding PUoW on that resource.

4.6.3 Conceptual Admissions Scenario

After choosing the LASM, the user must specify QoS preferences. Assume that the BF shown in Figure 7 is used for their example. For simplicity, because this is a simple benefit function, assume that the audio QoS is fixed and cannot be modified. The admissions scenario described here is conceptual; in practice, a more efficient algorithm must be used.

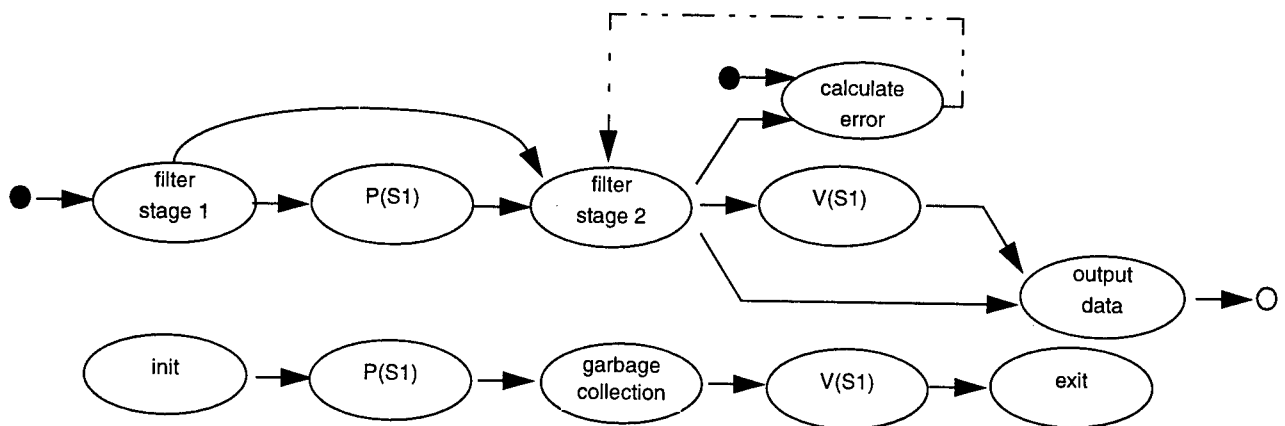
The system-wide resource management scheme first uses the BF to find the set of QoS values that maximize the benefit to the user. The system-wide resource management scheme then uses these values for the AIM. It then passes the LASM and the AIM to the allocation algorithm, which then determines a set of system resources to be allocated to this application, so as to meet the



(a) Video Teleconference LASM



(b) Two LRoSs for Video I/O Service



(c) One LRoSs for Audio Filter Service

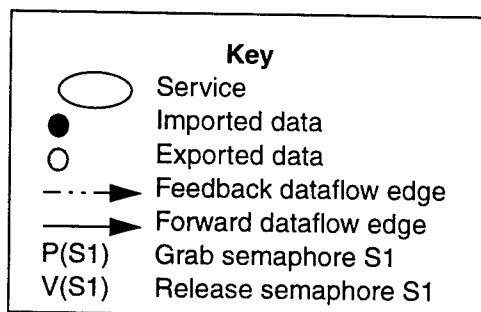


Figure 14. Example Logical Application Stream Model

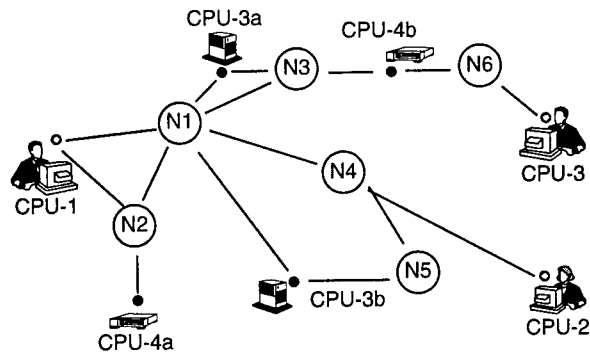


Figure 15. Example System

application's QoS requirements. If the allocation algorithm succeeds in this effort, the application is admitted. Otherwise, the system-wide resource management scheme finds the next highest benefit in the BF and inserts these QoS values into the AIM; it then reinvokes the allocation algorithm. This process continues until the allocation algorithm succeeds; if the possible combinations in the benefit function is exhausted without success by the allocation algorithm, the application is rejected. The next section describes how allocation may be done, given a set of QoS values using AIM.

Table 3. Example PUoWs*

| SERVICE | CPU 1 | CPU 2 | CPU 3 {a,b} | CPU 4 {a,b} |
|-------------------------|---------|---------|-------------|-------------|
| Sample background audio | SBA1 | | | |
| Sample audio | SA1 | | | |
| Fetch video | FV1 | | | |
| JPEG | JPEG1 | JPEG2 | JPEG3 | JPEG4 |
| JPEG ⁻¹ | UNJPEG1 | UNJPEG2 | UNJPEG3 | UNJPEG4 |
| MPEG | MPEG1 | MPEG2 | MPEG3 | MPEG4 |
| MPEG ⁻¹ | UNMPEG1 | UNMPEG2 | UNMPEG3 | UNMPEG4 |
| Filter stage 1 | FS11 | FS12 | | FS13 |
| Filter stage 2 | FS21 | FS22 | | FS23 |
| Init | Init1 | Init2 | | Init3 |
| Garbage collection | GC1 | GC2 | | GC3 |
| Output data | OD1 | OD2 | | OD3 |
| Exit | E1 | E2 | | E3 |
| Calculate error | CE1 | CE2 | | CE3 |
| Synchronize A/V | | SAV2 | | |
| Output audio | | OA2 | | |
| Display video | | DV2 | | |

*Note: Each row represents an LUoW from the LASM; each column represents a resource from the SM. Table entries represent the PUoW corresponding to the LUoW on the resource.

4.6.4 Conceptual Allocation Scenario

In this subsection we illustrate how our models can be used to achieve adaptive resource management. Assume that the source and destination are specified to be CPU 1 and CPU 2, respectively. These assumptions immediately constrain the location where the following services may execute: the sample audio, sample background audio, and fetch video services must be performed on CPU 1; the output audio and display video services must be performed on CPU 2.

The videoconference LASM uses eight child services to implement the videoconference service. For each of these eight services, the system-wide resource management scheme must choose a realization (an LRoS) from among all of the possible LRoSs and LUoWs. For the sake of simplification, we will assume that each LS other than the filter audio and video I/O in Figure 14(a) is realized only by a single LUoW. We also assume that each LS within the LRoSs given in Figures 14(b) and 14(c) are also realized by a single LUoW. Therefore, at the logical level, the system-wide resource management scheme really has two choices, namely, one of the two LRoSs for video I/O. The scheme can temporarily create two single-layer LASMs consisting entirely of LUoWs; in one, video I/O is realized via the JPEG LRoS, and in the other, video I/O is realized via the MPEG LRoS.

At the system-specific level, each LUoW can be realized on multiple resources, as shown in Table 3. Therefore, for these two temporary single-layer LASMs, we can create a set of temporary SASMs, each consisting of different combinations of PUoWs. Because some PUoWs cannot be used because of constraints, or because only a single PUoW exists, this approach prunes the search space. Table 3 indicates that the PUoWs SBA1, SA1, SAV2, OA2, and DV2 must be used.

Of the remaining PUoW combinations, the allocation algorithm tries each in order to determine its effect on application QoS and on system metrics. The algorithm chooses the PUoW combination that meets application QoS requirements and optimizes the system objective (according to the system metrics). (We do not describe this search algorithm here because it is beyond the scope of this report; Chatterjee [1997] describes such an integrated, QoS-based allocation and routing algorithm.) Figure 16 shows one possible outcome of the process described above.

As the system state changes, the allocation algorithm may find that a different combination of PUoWs results in a better system objective. If system resources fail, the allocation algorithm can re-execute this algorithm, but with the resource deleted from the system model, thereby finding a solution over noncompromised resources.

4.7 EXAMPLE TRANSLATION AND MAPPING

In this subsection, we describe the integration of the translation and mapping algorithms, focusing specifically upon the translation and mapping of applications onto network resources.

We begin with the following two definitions:

- **Translation** is the process of converting the QoS metrics of one protocol layer into the metrics appropriate at the adjacent higher or lower layer.
- **Mapping** is the process of correlating a set of QoS requirements with an identified set of system resources that will satisfy the user and application QoS requirements.

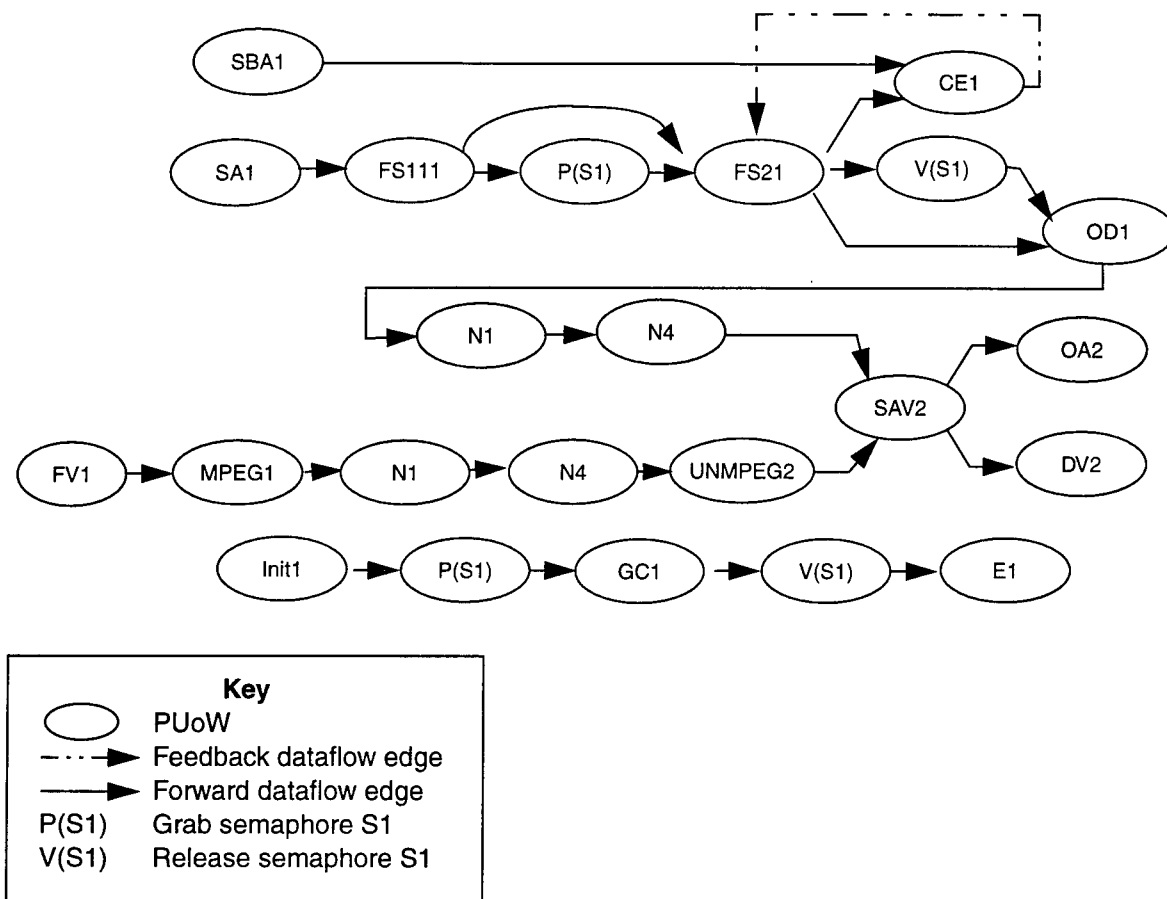


Figure 16. Composed Application

As shown in Figure 17, our concept of QoS translation and mapping includes vertical and horizontal perspectives. The vertical perspective provides for the translation of application QoS metrics down through communication protocol layers. Without loss of generality, a modified International Standards Organization (ISO) Open Systems Interconnection (OSI) reference model is used to define the communications protocol layers at end-system nodes and intermediate nodes (routers or gateways). For the end-system nodes (users), the protocol layers are the application, transport, network, data link, and physical layers. Intermediate system nodes (routers) consist of the network, data link, and physical layers. Within GRM, we define a QoS interface at the network layer. Thus, the choice of protocols employed below this network-layer interface depends upon the configuration of the underlying network technology in use, e.g., ATM, FDDI, or frame relay. The vertical translation process provides for the conversion of application QoS metrics to network-layer QoS parameters that are independent of the underlying subnetwork data link and physical layer communication protocols.

The convention used for naming the QoS parameters is as follows:

- QoS parameters of interest such as bandwidth, jitter, delay, loss ratio, and error ratio are represented as BW, J, D, LR, and ER, respectively.
- QoS parameters at each protocol layer contain a subscript that describes the layer. For example, the delay parameter at the transport layer is written as “D_{TRANSPORT}”.

- If the QoS parameters represent a value that is accumulated across the layers, an up or down arrow is used. If the direction of aggregation is from an upper layer to the layer below, a down arrow is used. The up arrow is used for aggregation in the opposite direction. For the example above, if the delay computed at the transport layer is based on the delay at the upper application layer (APPL), then the delay at the transport layer is represented by $D_{\text{TRANSPORT}\downarrow}$. If the delay computed at the transport layer is computed on the basis of delay at the network layer, then $D_{\text{TRANSPORT}\uparrow}$ is used to denote the cumulative delay at the transport layer.

The vertical perspective also provides for mapping these user and application QoS requirements into processing and storage resources at each protocol layer of the end-system node.

As shown in Figure 17, the horizontal perspective includes the mapping of network-layer QoS requirements, which are vertical QoS translations of application QoS requirements, and provides the mapping of these translated application QoS requirements to the resources of the heterogeneous communication network subsystem that connects the end-system nodes of interest. Representative network-layer QoS metrics include delay, bandwidth, jitter, loss ratio, and error ratio. The horizontal perspective also provides the translation of network-layer parameters to protocol-dependent subnetwork parameters, which can subsequently be used by system-wide resource management schemes of each subnetwork to perform admission control and to allocate or schedule resources.

4.7.1 QOS TRANSLATION AND RESOURCE MAPPING

4.7.1.1 Vertical QoS Mapping

As shown in Figure 17, vertical QoS mapping must account for the translation of user and application QoS requirements into QoS network-layer metrics, as well as mapping those requirements to resources.

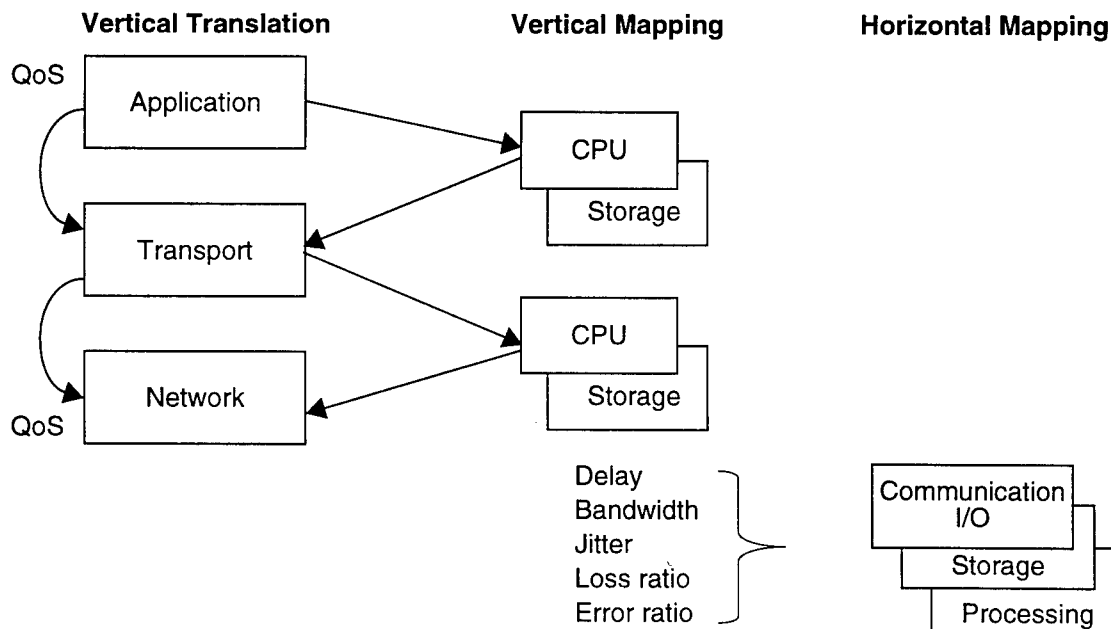


Figure 17. Vertical and Horizontal Translation and Mapping Perspectives

4.7.1.1.1 Vertical QoS Translation

The mathematics of the QoS translation process from the application layer down through the network layer are detailed in Section 6 and in Wang et al. [1997]. Conceptually, the process begins with the determination of the degree of satisfaction a user requires during the execution and presentation of information provided by an application. This level of satisfaction can be as clearly defined as the difference between a black-and-white and a color image, or it can be more subjective, depending on a user's ability to distinguish between video signals of different qualities. Quantifying user QoS metrics is largely an area for future study. In general terms, user QoS can be defined as excellent, good, or fair, and one of these levels of satisfaction would be required for each service provided by an application such as audio, video, or data services. The user QoS requirements for these application services would be defined in terms of time, precision, and accuracy attributes, but at a very high level that the user could understand. The user's requirements could be specified initially when the application program is first invoked, or they could be known *a priori* by the application and thus could be transparent to the user running the application. Once these levels of user satisfaction are determined, they are translated into application QoS requirements, which are more quantitative. Representative application QoS metrics are shown in Table 4.

For example, a user may specify that excellent video signal quality is required. At the application level, this requirement would translate to a precision metric of 30 video frames/s. A good-quality video signal might be represented by 20 frames/s. These values would then be translated into precision metrics for bandwidth at each of the underlying communication protocol layers, and eventually down to bits/s at the physical layer.

Table 4. Representative QoS Requirements

| Application Service | TIMELINESS | | PRECISION | | | ACCURACY |
|---------------------|--------------|----------------------|--------------------------|-----------------------|----------------------|----------------------|
| | Latency (ms) | Latency Variance (%) | Frame Size (bytes/frame) | Frame Rate (frames/s) | Frame Loss Ratio (%) | Frame Error Rate (%) |

The application QoS requirements of the different application services such as video, audio, or data services, must be derived from the desired performance of the entire application and not from the QoS requirements generally accepted for the individual services. For instance, the latency bounds for data services are generally allowed to be reasonably long, since readers are not interested in seeing a data file until the entire file is available. End-to-end network delays of minutes to an hour are rarely even noticed by e-mail readers. However, due to the near-real-time, interactive nature of collaborative multimedia, there must be a mutual consistency among the services in the application. For example, lip synchronization is needed between audio and video services, to ensure that video or audio services do not lead or lag behind the other service by any perceptible amount of time. Similarly, teleseminar end users want to see whiteboard data appear on their workstation screens while they are discussing the data; hence, the provision of data services must be consistent, just like video and audio, before the end users can perceive an application presentation as coherent. In general, mutual consistency among all time, precision, and accuracy QoS attributes is required in the provisioning of services that support near-real-time multimedia.

4.7.1.1.2 Vertical Resource Mapping

Within the vertical QoS mapping process, CPU processing and storage resources are provided at each protocol layer to support QoS translations. Processing functions include appending protocol headers and trailers, performing cyclic redundancy checks, and segmenting and reassembling protocol data units. Other types of processing requirements may dictate that specialized processors be used, which in a distributed processing environment may reside in a location external to the end system. In more general terms, processing resources are needed to provide, at each protocol layer, application data manipulation such as the transformation of the data to a different representation.

Storage resources include the memory space needed to store application and protocol code and data segments, and buffer space to provide bandwidth, delay and jitter guarantees. For example, storage resources would be required to support various windowing strategies used in networking protocols.

While processing and storage resources are matched to application QoS requirements at each protocol layer, communication or I/O resources are mapped at the network layer to application QoS requirements that have been translated down through the protocol layers. Once network-layer QoS metrics are identified at the completion of the vertical translation process within the source's end system, these metrics are mapped to communication resources provided by the network. A communication resource consists of a communication port, which provides I/O connectivity to communication links and channels that are specified by parameters such as bandwidth capacities, propagation delays, bit error rates, and other signal qualities. As with application QoS requirements, these specified communication resource parameters can also be categorized in terms of time, precision, and accuracy, which facilitates their mapping to QoS requirements.

4.7.1.2 Horizontal QoS Mapping

Once the application QoS requirements have been translated into network-layer QoS parameters, they are mapped to communication I/O resources for transport across the communication subsystem, as part of the horizontal mapping process.

As a result of the vertical translation process, QoS parameters generated at the network layer of the end-system node are defined in terms of bandwidth, delay, loss ratio, error ratio and jitter (as shown in Figure 17). The "Network-Layer" row of entries in Table 5 illustrates the QoS parameters that are derived from the corresponding application QoS requirements in the row above. Thus, the requirements for bandwidth at the network layer are derived from the frame size and frame rates supported by the user application. The network-layer QoS parameters are bandwidth ($BW_{NETWORK}$), delay ($D_{NETWORK\downarrow}$), jitter ($J_{NETWORK}$), loss ratio ($LR_{NETWORK}$), and error ratio ($ER_{NETWORK}$).

All of these network-layer QoS parameters have meaning within the end-to-end (ete) service connections provided among end users. In order to provide end-to-end QoS guarantees, the horizontal QoS mapping process translates the network-layer QoS parameters, which are communication-protocol independent, into QoS parameters that are communication-subsystem-protocol dependent for all the subnetworks along the connection's path within the communication subsystem.

Table 5. Network-Layer QoS Parameters in Relation to Application QoS Requirements

| VERTICAL LAYERS | COMMUNICATION RESOURCE PARAMETERS | | | | | |
|-----------------------------|-----------------------------------|-------------------------|-----------------------------|-----------------------|------------------------------|-------------------------------|
| | TIMELINESS | | PRECISION | | | ACCURACY |
| Application QoS Requirement | Latency (ms) D_{total} | Latency Variance (%) | Frame Size (bytes/frame) | Frame Rate (frames/s) | Frame Loss Ratio (%) | Frame Error Ratio (%) |
| Network Layer QoS Parameter | Delay $D_{NETWORK\downarrow}$ | Jitter $J_{NETWORK}$ | Bandwidth $BW_{NETWORK}$ | | Loss Ratio $LR_{NETWORK}$ | Error Ratio $ER_{NETWORK}$ |
| End-to-End QoS Parameter | D_{ete} | J_{ete} | BW_{ete} | | LR_{ete} | ER_{ete} |

These end-to-end QoS parameters can thus be equated to network-layer parameters by the following relationships:

$$BW_{ete} = BW_{NETWORK}$$

$$LR_{ete} = LR_{NETWORK}$$

$$ER_{ete} = ER_{NETWORK}$$

$$J_{ete} = J_{NETWORK}$$

Delay must be determined in a slightly different manner. The application QoS requirements for delay (D_{total}) must account for the total time required for processing and storage from one application layer to the destination end-station application layer. Thus, D_{total} is a function of the delay incurred through the vertical translation process at the originating end station, plus the total time needed to transport information across the communication network subsystem, and the time to allow for the vertical translation process at the destination end station. To be consistent with the notation by Wang et al. [1997] and Section 7, D_{total} is represented by the following relationship:

$$D_{total} = D_{NETWORK\downarrow} + D_{ete} + D_{APPLICATION\uparrow}$$

Parameters such as $LR_{NETWORK}$ and $ER_{NETWORK}$ that are translated from the application QoS requirements are compared with the LR_{ete} and ER_{ete} , which are computed as functions of protocol-specific subnetwork parameters. For example, the network layer's bandwidth and loss ratio are mapped to the peak or average cell rate and cell loss ratio, respectively, within an ATM subnetwork. ATM cell delay is then a function of D_{ete} . This translation process assists the system-wide resource management schemes of each subnetwork in performing admission control and allocating resources.

Horizontal mapping also involves the mapping of end-to-end QoS requirements to the resources required of the communication subsystem, along the horizontal direction. These requirements include the processing and buffer space needed from the intermediate nodes to perform packet routing and absorb the jitter caused by variations in processing and queuing delay, and the bandwidth required across adjacent subnetworks along the end-to-end path. The techniques for further deriving these parameters are discussed in Section 6 of this report.

4.7.2 CONCLUSIONS

In this section we have provided an approach to the mapping of application QoS requirements to resources, in terms of processing, storage, and communication subsystems. (The vertical and horizontal translation and mapping processes are illustrated in Figure 17.)

In this approach, user applications communicate over a heterogeneous communication subsystem through a protocol stack that can be generically described by a modified OSI reference model. The point of interface for an application's QoS requirements and the underlying communication network subsystem resides at the network layer, where application QoS can be described in communication-protocol-independent terms. Processing resources are mapped according to the CPU speed and the types of processing that are required at each layer of the end-system nodes. Storage resources are mapped based on the amount of buffer space needed at end-system nodes and intermediate nodes along an end-to-end connection, in order to provide end-to-end jitter, bandwidth, and delay guarantees. Communication resources are computed according to an application's bandwidth requirements and expressed in parameters specific to the protocols employed by the subnetworks along the end-to-end connection.

5 CORBA-BASED RESOURCE MANAGEMENT STRUCTURE

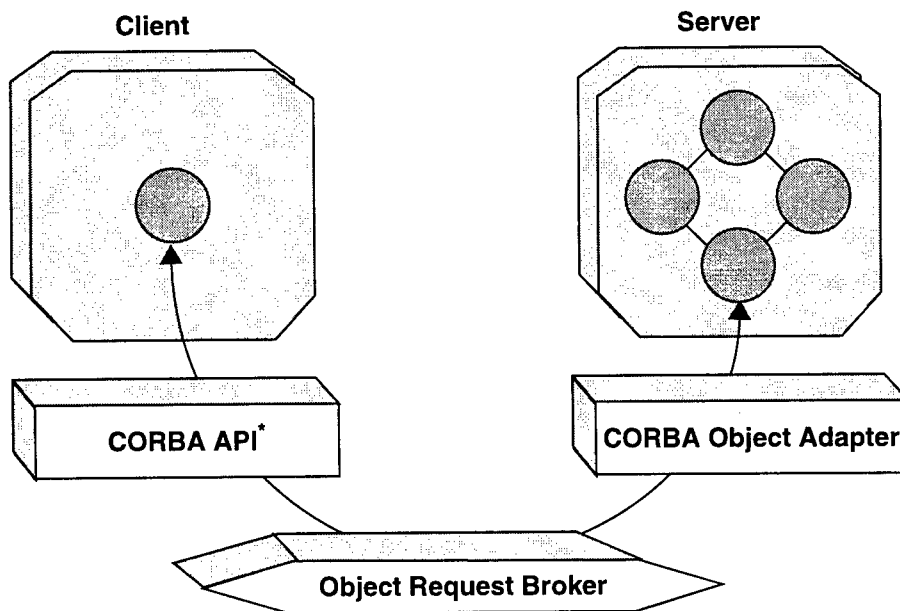
5.1 OVERVIEW OF CORBA

The Common Object Request Broker Architecture is a middleware standard for distributed object computing, developed by Object Management Group, Inc. (OMG) and X/Open Co. Ltd., and endorsed by several organizations such as International Business Machines, Inc. (IBM), Digital Equipment Corporation, Hewlett-Packard Company, Hyperdisk Corporation, NCR Corporation, Object-Design Inc., and SunSoft Inc. The CORBA model has been proposed to support the implementation of operation-independent, protocol-independent, object-oriented interfaces.

CORBA supports flexible and reusable distributed services and applications, using *Object Request Brokers* (ORBs). An ORB can translate object requests and operations between different management networks. All object requests made in CORBA by a *Client* are sent to a *Server* and are handled by an ORB that serves as an interface between the two objects. Thus, the ORB provides the communication and activation infrastructure for distributed object applications.

Key components of CORBA are illustrated in Figure 18. Figure 19 illustrates the use of the ORB for client-server requests. They are defined as follows.

- ORB. Enables objects to transparently make and receive requests and responses in a distributed environment. The ORB connects objects that request services or functions with objects that can satisfy the requests. Applications do not need to know the locations of the remote objects. The ORB locates appropriate object implementation (code) and passes parameters and control between a client and a server.



*API: Application Programmer's Interface.

Figure 18. CORBA Organization and Components

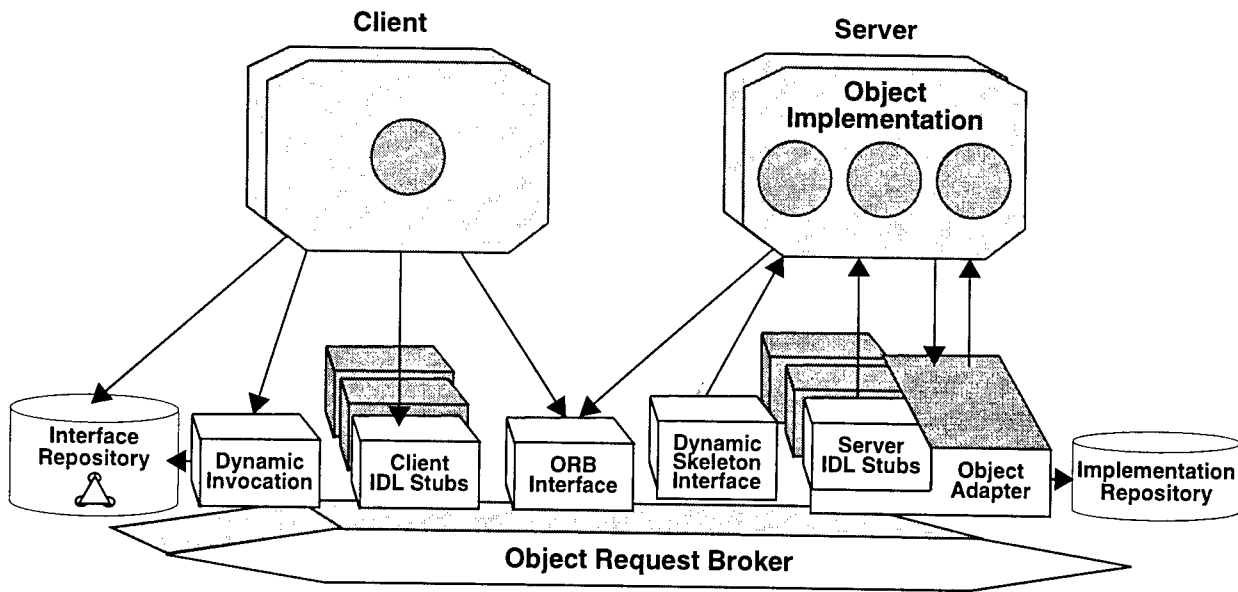


Figure 19. CORBA Client-Server Request

- **Interface Definition Language (IDL).** Enables a particular object implementation to tell its clients what operations are provided and how they could be invoked. IDL is independent of any programming language. IDL is used to define the types of objects by specifying the interfaces for each object. Interfaces include a set of operations and the parameters for those operations.
- **IDL Stubs.** Static means by which clients invoke operations on objects.
- **General Inter-ORB Protocol (GIOP).** Allows for communication between ORBs. GIOP is responsible for ensuring a common data representation and enabling message transfer within GIOP-defined message formats.

5.2 FEASIBILITY OF A CORBA-BASED RESOURCE MANAGEMENT STRUCTURE

In general, the objectives of CORBA are very much in line with the requirements of GRM. Because CORBA is structured to allow the integration of a wide variety of object-oriented systems, it is well suited to a heterogeneous and distributed enterprise network. Transparency between network boundaries is facilitated by CORBA. The client, which can be one component of the system model, does not need to know about the other components of the model. Specifics concerning the destination network, the location of the services in the application flow, implementation details such as the programming language of a LuOW or the hardware resources the service uses, and the current state of the service are hidden from the client by the ORB.

The specific tasks required of the global resource manager can be generally classified as QoS-based service provisioning or resource management. Service provisioning, with the support of QoS specification and translation, supports QoS guarantees through admission control, QoS negotiation, resource allocation, and QoS trade-off. Effective resource management includes anomaly management, monitoring and control, and traffic prediction.

CORBA-Based Service Provisioning. Although QoS specification and translation are not currently supported by CORBA, a CORBA service has been introduced that may be used to facilitate QoS-based resource management. The Trader service enables clients to obtain object references for target objects by specifying the properties of the objects. If the CORBA standard is modified to enable servers to specify their QoS parameters, then the servers could use CORBA's Trader service to automatically select the child services that best fit their requests by specifying the desired network-level QoS. Extensions to CORBA, either through IDL or in the form of additional services, will be required for the support of QoS specification and for QoS mapping and translation.

CORBA-Based Management. CORBA can be used to maintain a logical link between the various managers of enterprise subsystems, regardless of legacy management protocol. By means of management domain translation, CORBA can also be used to manage the SNMP and CMIP agents within the enterprise subsystems, facilitating end-to-end integrated management.

A static method for management domain translation is very near completion by the Joint Inter-Domain Management Task Force (JIDM), a joint X/Open-NMF* task force that is examining the issues of interworking between CORBA and CMIP/SNMP.[†] The JIDM algorithm is performed in two stages: the translation of the SNMP/CMIP management information bases (MIBs) to CORBA IDL on the client side, and the translation of the CORBA invocation to an SNMP/CMIP request/response on the server side. These stages are called Specification Translation and Interaction Translation, respectively.

Because CORBA requires the use of IDL in the definition of network management system APIs, the client-side translation involves converting the structure of the MIB into IDL format. The JIDM has developed Specification Translation algorithms to translate CMIP GDMO[‡] or SNMP MIBs to IDL.

The server-side translation requires a conversion of CORBA management operations to SNMP/CMIP PDUs. Specifically, the CORBA invocation that was issued in response to the client's request must be converted to the appropriate SNMP/CMIP request or response, such as Get or GetResponse. The JIDM is currently developing Interaction Translation algorithms for translating CORBA invocations to CMIP or SNMP requests.

These translations are, in effect, generic gateways between the technologies, operating on a run-time basis and dealing with the way that CORBA operation invocations should be translated into CMIP or SNMP requests. In accordance with the concept of CMIP/SNMP object adapters on the server side, the Interaction Translation algorithms will use the CORBA Dynamic Skeleton Interface (DSI) and the Dynamic Implementation Routine (DIR) at run time on the server side.

Therefore, CORBA offers many features to support the convergence of resource management toward a distributed, homogeneous, global management architecture.

*NMF: Network Management Forum. All product or company names mentioned in this document are the trademarks of their respective holders.

[†]CMIP: Common Management Information Protocol; SNMP: Simple Network Management Protocol.

[‡]GDMO: Guidelines for the Definition of Managed Objects.

5.3 APPLICATION OF CORBA CONCEPTS TO GRM MODELS

Because the GRM models have been developed within an object-oriented framework, CORBA concepts map easily into the GRM structure.

Application Model. The GRM Application Model describes the application flows as a decomposition of services. This decomposition can be implemented by considering parent services as clients decomposing into logical child services, which would be CORBA objects. Application invocations are instantiations of the logical application services.

The application model is recursive, because a CORBA client in one CORBA operation can be a CORBA object in another operation. The application is represented as a client making a request to a number of LRoSs (CORBA objects) that could perform the needed services. These LRoSs in turn become clients in search of lower-level LRoS or PLoS CORBA objects. Thus, in the decomposition of an application flow, the application is always a client and the PLoS is always a CORBA object. The LRoS can be either a client or an object, depending on whether it is responding to a request from a parent LRoS (CORBA object), or decomposing further (client).

System Model. The GRM System Model features subsystems of subsystems with varying domains of control. A system database resides in each subsystem, which stores information pertaining to the services available in the subsystem, the resources within the subsystem, and the QoS capabilities of the subsystem. This database corresponds to CORBA's Implementation Repository.

In the System Model, services that support an application can be located in separate subsystems. CORBA supports this distribution with the use of standard CORBA operations, where the caller is blocked until a response is received from the object. Thus, the resource manager at the highest level, acting as a client, makes a service request to the manager (CORBA object) of the preferred subsystem. This manager determines whether it can fulfill the request; if it can, acting as a client it invokes a request for services within the subsystem. If it cannot fulfill the request, it responds negatively to the client, which then searches for other child subsystems to perform the service.

Resource Model. The GRM Resource Model describes the physical resources that are mapped to PLoSs of the Application Model. Thus, the Resource Model is used by the subsystem manager to select the appropriate PLoS.

5.4 CHALLENGES

Despite the benefits that CORBA offers GRM, several challenges remain in effecting end-to-end QoS for real-time services. The performance levels and QoS enforcement features of current CORBA implementations are not yet suited to hard real time systems (e.g., avionics) and constrained latency systems (e.g., teleconferencing).

Fault Tolerance. Contemporary CORBA ORBs use communication support provided by operating systems (i.e., sockets). Therefore, predicting the behavior of distributed applications implemented over these ORBs, in the event of partial failures, is difficult or impossible. Considering the importance of fault tolerance and reliability to the GRM system, this problem would be a serious detriment.

Lack of Standard QoS Policies and Mechanisms in CORBA. QoS specification must be supported both per request and per object. However, the CORBA specification does not currently define policies or mechanisms for QoS specification or for providing end-to-end QoS guarantees. There is no standard means of specifying performance, security, priority, or cost requirements through CORBA, and there is no means of specifying to the ORB the timing or other relationships that may be required between invocations of objects [Schmidt et al. 1997a]. However, the OMG has assigned a task force to investigate this problem.

Lack of Real-Time Features in CORBA. CORBA does not currently define a standard way to invoke CORBA requests asynchronously. In addition, CORBA does not define a threading model, and does not require the ORB to notify clients when transport-layer flow control occurs. Without these capabilities, a standard CORBA application could block indefinitely as a result of constrained resources [Schmidt et al 1997a].

Lack of Performance Optimizations in CORBA. Existing ORBs incur significant overhead, reducing throughput and increasing latency.

5.5 RELATED WORK

Extensions of the CORBA model are being investigated to address some of the challenges in implementing a CORBA-based resource management system for real-time multimedia applications.

Enhancing Reliability. Maffeis and Schmidt [1997] at Washington University have considered the improvements in fault tolerance and reliability that can be made possible by extending the CORBA model to support the Virtual Synchrony model proposed for the ISIS toolkit, which is a run-time technology for reliable distributed computing. This can be achieved by layering the ORB on top of the ISIS toolkit, rather than building directly on sockets. Virtual Synchrony supports object group computing, which ensures high availability through process replication. Maffeis and Schmidt [1997] further propose to integrate this architecture with transaction processing (TP) monitors and message queues in order to realize the combined benefits of these technologies.

TP monitors allow a distributed client application to bracket a series of service invocations with begin/end transaction markers, and can thus roll back invocations issued within the transaction. TP monitors can therefore be used to maintain distributed data consistency, despite crash failures, by employing this rollback mechanism. TP monitors have been in use for several years in support of mission-critical banking applications, and have recently been standardized by the OMG to be used in conjunction with CORBA applications. Orbix+Tuxedo offers such an integration.

Message queues facilitate the decoupling of sender processes from receiver processes, allowing a sender to submit a message without knowing whether the receiver is available. Message queues have also been in use in the financial domain for many years. The OMG has recently standardized an Event Channel service specification to be used in conjunction with CORBA applications (an example is OrbixTalk, which distributes requests via IP multicast) in order to implement message queues within the CORBA model.

Performance Optimizations to Support Real-Time CORBA. Schmidt et al. [1997] have also considered the requirements of the end system architecture for real-time CORBA and mechanisms to satisfy these requirements. The components of this architecture include the following:

- A gigabit I/O subsystem to optimize conventional OS I/O subsystems, enabling real-time scheduling of OS and network resources and providing efficient buffer management that shares client request buffers across OS protection domains
- A suite of real-time GIOP/IOP protocols to support the predictable and efficient transmission of requests (customized for specific application requirements)
- A real-time object adapter to schedule and dispatch CORBA requests
- Application-specific components to provide features that support end-to-end QoS guarantees
- A set of real-time IDL (RIDL) schemas that enable applications to specify QoS attributes for their object's operation
- An off-line Scheduling Service that determines the priority and scheduling characteristics of client requests with hard real time deadlines
- Real-time Event Channels that use the real time ORB to support customized scheduling, concurrency, and filtering policies for user-defined events.

Quality of Service in CORBA. Zinky, Bakken, and Schantz [1974] have developed an architecture, Quality of Service for CORBA Objects (QuO), to provide QoS abstractions to CORBA objects. QuO extends the CORBA functional IDL with a QoS Description Language (QDL). QDL is intended to specify an application's expected usage patterns and QoS requirements for a connection to an object. Thus, QDL allows the object designer to specify system states, which represent the status of a negotiated QoS contract. The system states then provide the necessary information to measure and enforce QoS contracts.

CORBA Extensions in Support of Real-Time Applications. In response to an OMG Request for Information, Wolfe and Johnson [1997] have developed real-time extensions to CORBA, enabling the support of end-to-end QoS specification and guarantees, policing, and priority-based scheduling. These extensions include the following:

- Real-time Manager, implemented as a C++ class with most of the capabilities for Timed Distribution Method Invocations
- Real-time Environment, defined and specified in IDL
- Real-time Exceptions, which are generated within the CORBA exception mechanism in order to alert the manager to violations of QoS constraints specified in the Real-time Environment
- Global Time Service for all objects in the CORBA environment
- Global Priority Service, which translates QoS specifications, importance specifications, and age to a single priority number
- Distributed Real-time Scheduling, which is a modification to a commercial, off-the-shelf (COTS) ORB that requires priority-based scheduling and queuing
- Real-time Event Service, extending CORBA 2.0 Event Service by allowing events to indicate the times at which they occurred

- Real-time Concurrency Control Service, extending the CORBA 2.0 Concurrency Control Service by allowing priority inheritance
- Network Latency Service, which enables high-confidence estimates of message latency in the CORBA system
- Real-time bind, to find the appropriate server on which to bind a client based on QoS specifications.

Much of the work being performed in these areas can be applied to the implementation of GRM over CORBA, particularly the work that is performed in concert with the OMG.

5.6 AREAS OF FUTURE WORK

If a CORBA-based resource management system is implemented, extensions to the current CORBA framework will be necessary, particularly in the following areas: QoS specification and guarantees, application-specific ORB communication protocols, and better support of real-time applications through additional CORBA services and ORB performance optimizations to reduce latencies. Future CORBA specifications will, however, likely address many of the extensions necessary to support real-time applications, including a standard, CORBA-based approach to real-time scheduling, timing constraint specification, and priority-based queuing for concurrency control. The areas of particular interest to GRM are therefore those related to CORBA support of QoS.

A CORBA service is required that will perform QoS mapping and translation, as described in Section 4. As an object service, QoS mapping can be implemented by an object.

The CORBA standard must be amended to allow the client's priority and QoS requirements to be attached to its method invocation request. Likewise, the CORBA standard must allow for the specification of QoS parameters by the servers. This information must be available to the ORB, CORBA services, dynamic skeleton interface, and server implementations. In addition, the CORBA exception mechanism should be extended for the incorporation of additional CORBA exceptions needed to flag instances when QoS guarantees have not been met.

The CORBA environment should support "performance polymorphism" in order to provide ORB-controlled QoS guarantees [Wolfe and Johnston 1997]. Performance polymorphism extends the object-oriented notion of polymorphism and dynamic binding, which typically allows the run-time selection from among many possibilities of the appropriate method to execute. The potential methods are usually invoked on the same object interface, which is often arrived at through inheritance. In performance polymorphism, the selection of the appropriate method is made based on either the allowed QoS or the requested QoS.

Although CORBA is not ready to provide the middleware to GRM, the inherent benefits CORBA offers, as well as the interest expressed by the OMG in the incorporation of real-time capabilities, make a CORBA-based architecture worth pursuing.

6 RESOURCE MANAGEMENT FOR TELESEMINAR

An example teleseminar application consisting of video, audio, and whiteboard services illustrates the concepts described in this section. The section is organized as follows: Subsection 6.1 briefly describes end-to-end user QoS requirements and the translation of user requirements to application requirements; Subsection 6.2 describes the decomposition of a teleseminar application service via the Application Model; Subsection 6.3 describes an example heterogeneous system where the teleseminar application service resides; and finally Subsection 6.4 illustrates the mapping of the QoS requirements associated with each LRoS and PRoS to resources, in order to provide end-to-end application QoS guarantees.

6.1 USER QOS REQUIREMENTS

For simplicity, we define qualitative user QoS parameters as *excellent*, *good*, *fair*, or *poor*. Tables 6, 7, and 8 summarize the translation of QoS from user to application requirements for the video, audio and whiteboard services, respectively, of the teleseminar application. For the whiteboard service, a maximum burst rate of 10,000 bytes/s is assumed, and 5,000 bytes/frame is used as the maximum frame size.

Within each table, quantitative QoS parameters are grouped into the QoS metric categories time, precision, and accuracy. Table 6 illustrates a translation of user QoS to application QoS for a video service using MPEG compression.

Table 6. Translation of User QoS to Application Video QoS

| USER QOS | TIMELINESS | | PRECISION | | | ACCURACY |
|-----------|-------------------------|----------------------|--------------------------|-----------------------|----------------------|-----------------------|
| | END-TO-END LATENCY (MS) | LATENCY VARIANCE (%) | FRAME SIZE (BYTES/FRAME) | FRAME RATE (FRAMES/S) | FRAME LOSS RATIO (%) | FRAME ERROR RATIO (%) |
| Excellent | 200 | 2.5 | 31104 | 25–30 | 0 | 1 |
| Good | 350 | 5 | 20736 | 15–24 | 2 | 5 |
| Fair | 500 | 10 | 1556 | 7–14 | 5 | 10 |
| Poor | 650 | 20 | 150 | 6–14 | 10 | 20 |

Tables 7 and 8 illustrate the translation of user to application QoS for the audio services, and of user to application whiteboard QoS.

Table 7. Translation of User QoS to Application Audio QoS

| USER QOS | TIMELINESS | | PRECISION | | | ACCURACY |
|-----------|-------------------------|----------------------|---------------------------|-------------------------|-----------------------|------------------------|
| | END-TO-END LATENCY (MS) | LATENCY VARIANCE (%) | SAMPLE SIZE (BITS/SAMPLE) | SAMPLE RATE (SAMPLES/S) | SAMPLE LOSS RATIO (%) | SAMPLE ERROR RATIO (%) |
| Excellent | 200 | 2.5 | 32 | 44,100 | 2 | 5 |
| Good | 350 | 5 | 8 | 8,000 | 10 | 10 |
| Fair | 500 | 10 | 2 | 8,000 | 20 | 20 |
| Poor | 650 | 20 | 2 | 8,000 | 30 | 30 |

Table 8. Translation of User QoS to Application Whiteboard QoS

| USER QOS | TIMELINESS | | PRECISION | ACCURACY |
|-----------|-------------------------|----------------------|---------------------|----------------------|
| | END-TO-END LATENCY (MS) | LATENCY VARIANCE (%) | DATA LOSS RATIO (%) | DATA ERROR RATIO (%) |
| Excellent | 200 | 2.5 | 2 | 5 |
| Good | 350 | 5 | 10 | 10 |
| Fair | 500 | 10 | 20 | 20 |
| Poor | 650 | 20 | 30 | 30 |

6.2 APPLICATION MODEL

As described in Section 3, we can view an application as a service that an application user requests from the system. This service can, in turn, request other services from the system on its own behalf. Therefore, each parent service can be described as being composed of a set of child services executing in an ordered manner. Each child service can, in turn, be described as being composed of its own set of child services executing in an ordered manner. This recursive decomposition can continue until an application is defined by a set of units of work that are not composed of additional services and can also specify the load they place on a single computing, storage, or communication resource. In this section, we will use the teleseminar application as an example to illustrate how to recursively decompose an application into UoWs.

The teleseminar application can be viewed as a service that in turn comprises video, audio, and whiteboard services. The video and audio services provide continuous data streams, while the whiteboard service generates data in a random and bursty fashion. Figure 20 illustrates the decomposition of the teleseminar service.

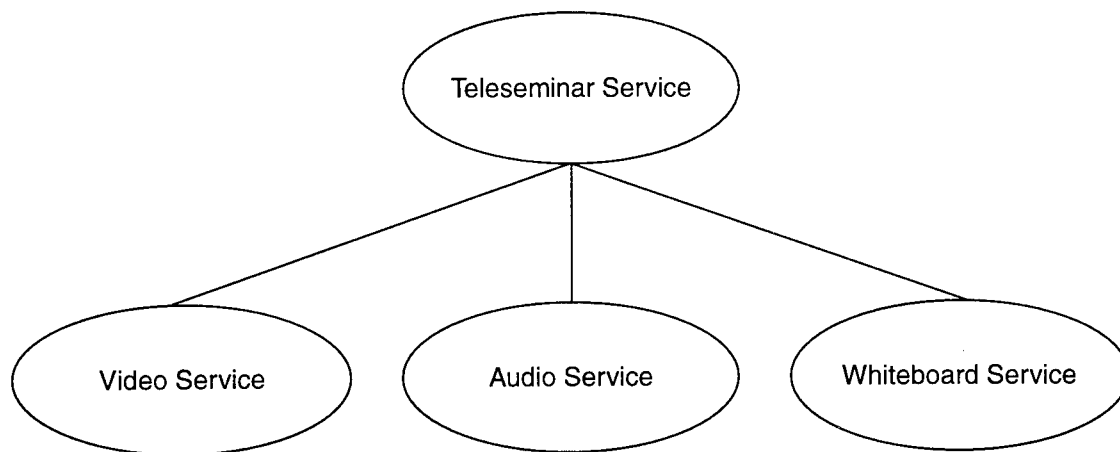


Figure 20. Decomposition of the Teleseminar Service

We now focus on the decomposition of the video service. The video service can be further decomposed into fetch video service, video I/O service, and display video service. Among these child services, fetch video service and display video service can be viewed as P_{Ro}Ss, while video I/O service can be represented as an L_RoS. Figure 21 illustrates this decomposition.

For illustrative purposes, we will assume that the video I/O service is implemented in the following steps:

- Transmit video frame from the source machine to the MPEG compression server.
- Execute the MPEG compression algorithm.
- Transmit the compressed video frame from the MPEG compression server across the enterprise network to the MPEG decompression server.
- Execute the MPEG decompression algorithm.
- Transmit the decompressed video frame from the MPEG decompression server to the destination workstation.

Figure 22 depicts the decomposition of the video I/O service based on the above assumption.

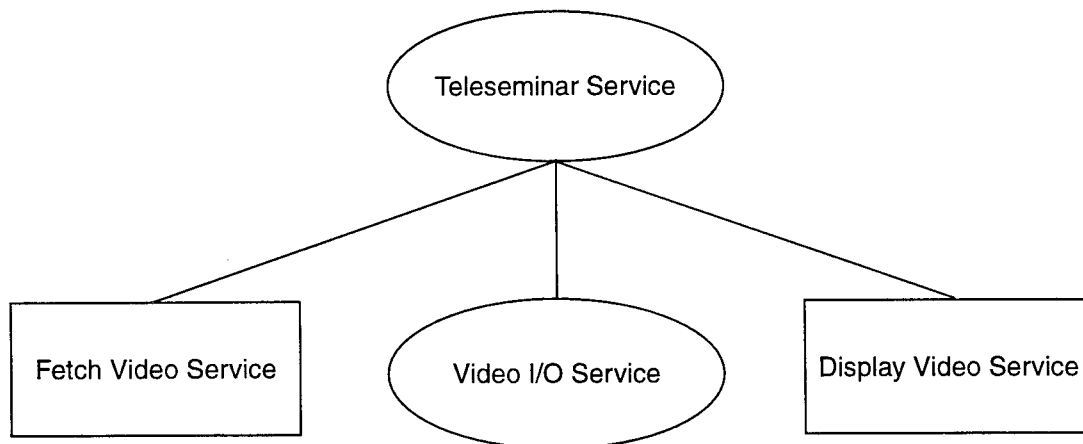


Figure 21. Decomposition of the Video Service

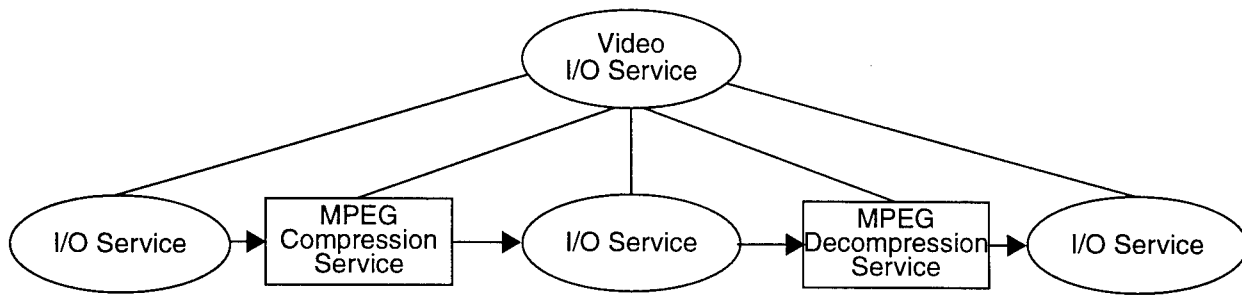


Figure 22. Decomposition of the Video I/O Service

Continuing in the same fashion, the video service can be recursively decomposed into UoWs, as illustrated in Figure 23.

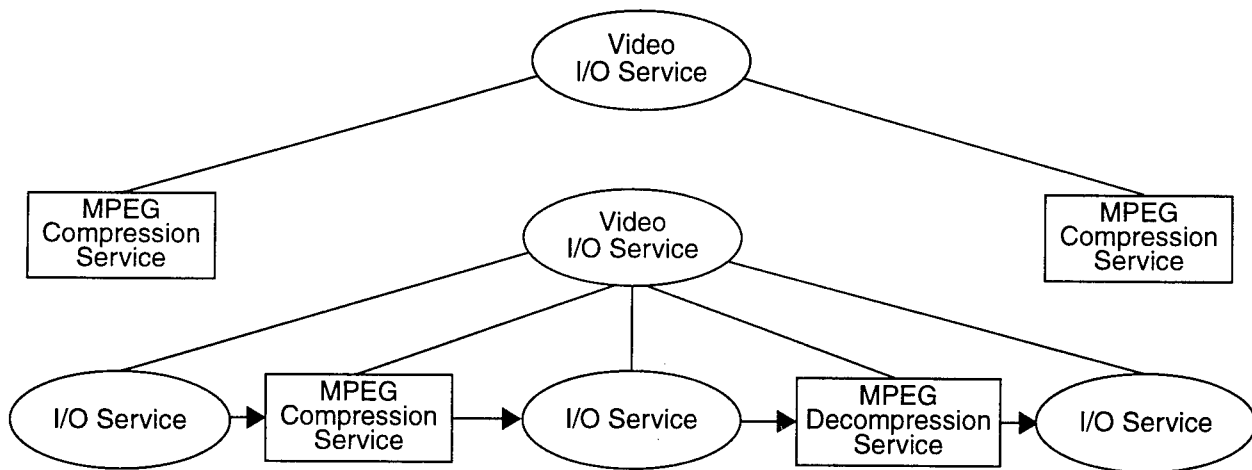


Figure 23. Video Service Decomposition

6.3 SYSTEM MODEL

In this subsection, we use the System Model described in Section 3 to model an example system where the teleseminar application resides. The example system consists of one global system, S0, at the top of the SM hierarchy. This global system consists of several subsystems that are interconnected by the Internet. Assume that the teleseminar application resides in one of the subsystems, SS1, and that SS1 itself has three subsystems named SS10, SS11, and SS12. Subsystem SS11 can be further decomposed into four subsystems: SC0, SC1, SC2, and SC3, each having one resource, C0, C1, C2, and C3, respectively. C0 represents an ATM network, C1 represents a workstation on which a user can pass the teleseminar application request, C2 represents an arbitrary file server, and C3 represents a MPEG compression server. Similarly, subsystem SS12 consists of four subsystems, SB0, SB1, SB2, and SB3, with the resources B0, B1, B2, and B3, respectively. B0 represents an Ethernet LAN, B1 represents a server on which an audio

filter application runs, B2 represents a MPEG decompression server, and B3 represents the destination workstation for the teleseminar application. Finally, subsystem SS10 consists of one router interconnecting the ATM network and the Ethernet LAN. Figure 24 illustrates the example system described above.

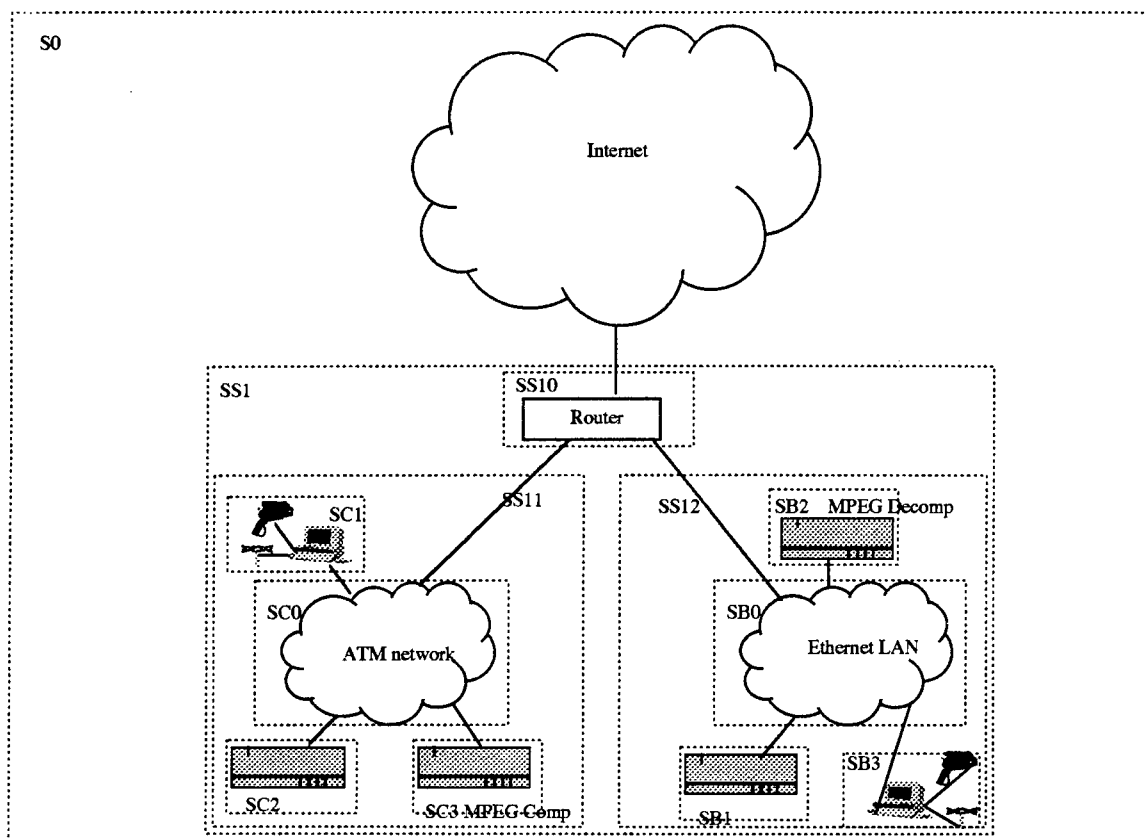


Figure 24. Example System for Teleseminar Application

6.4 QOS MAPPING

In order to provide end-to-end QoS guarantees, QoS translation is applied to LRoSs and PRoSs. QoS translation helps resource managers to choose optimized LRoSs and PRoSs, and to perform admission control. For example, translation of the delay QoS parameters shows how the resource manager can be assisted in performing admission control. Assuming that the system database for the subsystem of interest contains one LRoS for video service, and that the resource manager within that subsystem knows that the delay QoS parameter for this LRoS is 150 ms, the resource manager needs to determine if this LRoS can satisfy the delay QoS requirement. Figure 26 shows how the video service can be finally decomposed into PRoSs and I/O LRoSs. The delay for the parent LRoS is calculated by summing the delays of its child services (either PRoSs or LRoSs).

Assuming that each PRoS has an associated delay of 5 ms and each I/O LRoS has an associated delay of 50 ms, the video I/O LRoS has a delay of 160 ms, resulting in a delay of 170 ms for the video service LRoS. Because this computed delay exceeds the 150 ms delay requirement, the admission control test fails. Figure 25 illustrates this example.

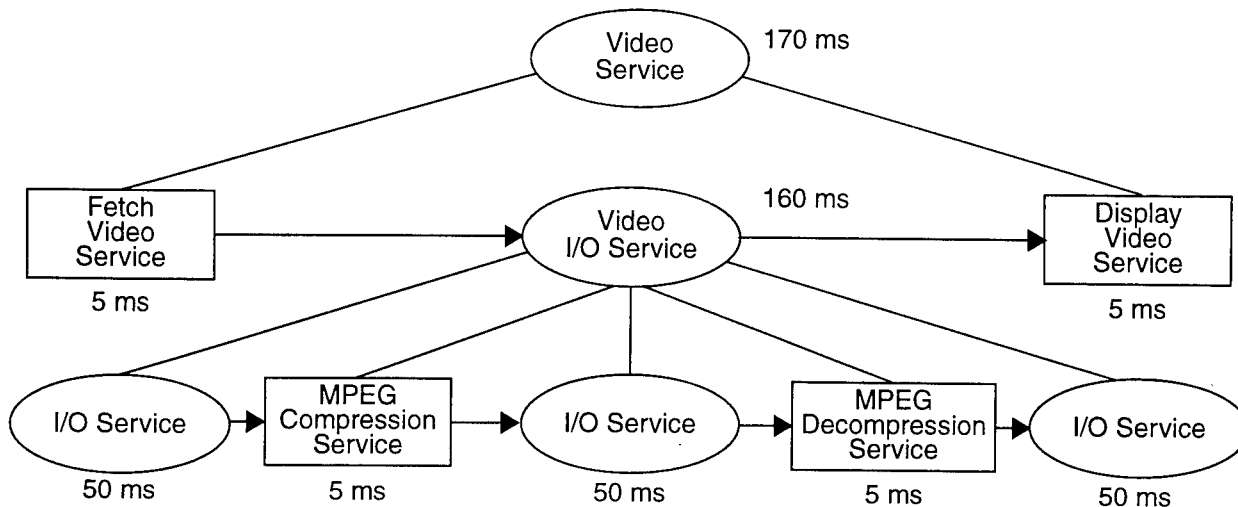


Figure 25. Delay Aggregation Example of QoS Translation

QoS mapping is applied to both PRoSs and I/O LRoSs in order to map QoS requirements into the resources used by resource managers to perform resource allocation. The resource mapping used in this section uses both vertical and horizontal QoS mapping approaches.

As mentioned in Section 4, vertical QoS mapping involves layer-by-layer QoS translation and resource mapping. The resources of interest are (1) the processing time associated with the data manipulation incurred at each of the protocol layers of the end system nodes, and (2) the buffer space required at the network layer for absorbing jitter and providing loss-free guarantees. The results of vertical QoS translation are used by horizontal QoS mapping. Therefore, vertical QoS mapping can be applied to PRoSs.

The QoS translation involved in horizontal QoS mapping converts the end-to-end QoS parameters generated by the vertical QoS translation into protocol-specific QoS parameters that can be understood and used by the resource managers within the underlying subnetworks to perform resource allocation. For resource mapping, the resources of interest are (1) the communication bandwidth for each of the subnetworks along the connection's path, (2) the time spent on processing, such as the time expenditure for packet routing incurred in the intermediate routers that interconnect the subnetworks along the path, and (3) the buffer space required at the network layer of the intermediate routers to absorb jitter and provide loss-free guarantees. Thus, horizontal QoS mapping can be applied to I/O LRoSs.

As shown in Figure 26, a video service is decomposed into a fetch video PRoS, an I/O LRoS, an MPEG compression PRoS, an I/O LRoS, an MPEG decompression PRoS, an I/O LRoS, and a display video PRoS. In this example, vertical QoS mapping is applied to the fetch video PRoS. The results of the vertical QoS translation are used by the horizontal QoS mapping service; this service is applied to the I/O LRoS that transmits video data from the source machine where the fetch video PRoS resides to the MPEG compression server. Again, vertical QoS mapping is applied

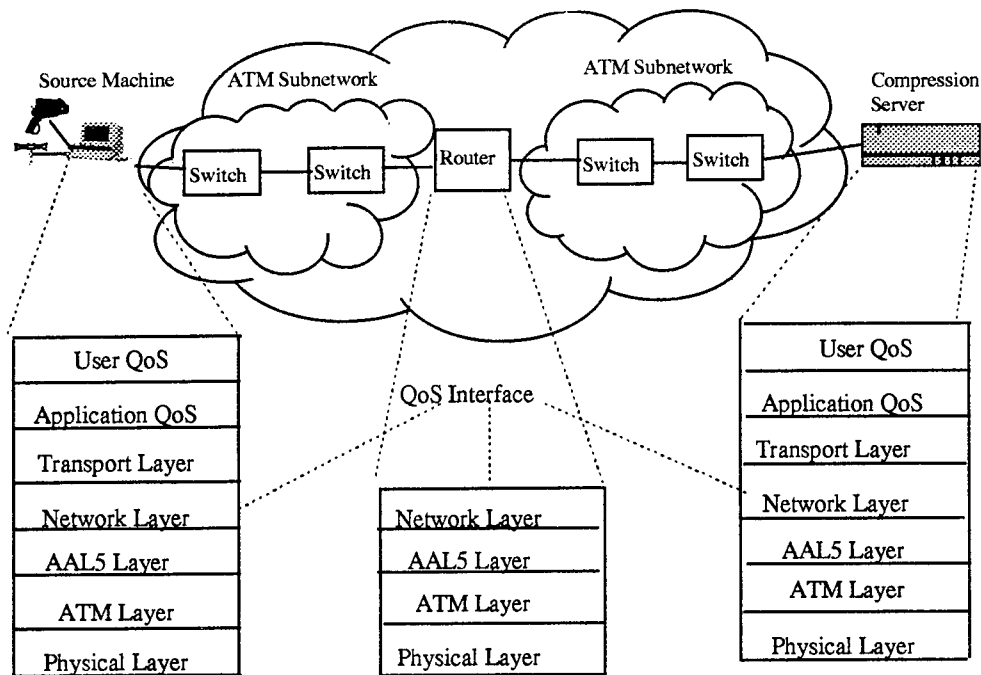


Figure 26. Network Configuration Used for QoS Mapping

to the MPEG compression PRoS, and horizontal QoS mapping is applied to the I/O LRoS, which transmits the compressed video data from the MPEG compression server to the MPEG decompression server across the network. Continuing in the same fashion, vertical or horizontal QoS mapping is applied to all the PRoSs and I/O LRoSs; thus, the application QoS requirements for the video service are mapped into the resources required by each PRoS and I/O LRoS.

In the following subsections, we use the MPEG compression PRoS as an example of vertical QoS mapping; and we use the I/O LRoS that transmits video data from the source machine to the compression server as an example of horizontal QoS mapping. As depicted in Figure 24, an ATM network interconnects the source machine and the compression server. Refer also to Figure 26, which depicts an example of the ATM network configuration used for QoS mapping.

6.4.1 Vertical QoS Mapping for MPEG Compression PRoS

When a video frame arrives at the network layer of the MPEG compression server, it is passed up to the application layer, where the compression algorithm is performed. Finally, the compressed video frame is passed down from the application layer to the network layer for transmission across the network. Thus, QoS translation contains two phases: network layer to application layer (before compression), and application layer to network layer (after compression).

6.4.1.1 QoS Translation

The QoS parameters of interest for the QoS translation process are bandwidth and delay, assuming that compression is loss and error free.

Bandwidth. For the bandwidth QoS parameter, the two phases of QoS translation are similar. Therefore, only the QoS translation process for the second phase is described in this subsection. However, the translation results for both phases are given in a numerical example. The relationship between the compressed and uncompressed video sizes as follows:

$$DS_{APPL-COMPRESSED} = DS_{APPL-UNCOMPRESSED} / CR$$

where CR is the compression ratio and DS represents the data unit size (in this case, the frame size).

Assuming that BW_{APPL} (the frame rate) is not changed after compression, the bandwidth requirement at each layer can be derived as follows:

Application Layer to Transport Layer

$$BW_{TRANSPORT} = BW_{APPL} \text{ (packets/s)}$$

$$DS_{TRANSPORT} = DS_{APPL-COMPRESSED} + HS_{TRANSPORT} \text{ (bytes/packet),}$$

where $HS_{TRANSPORT}$ is the number of bytes in the header of a transport layer packet.

Transport Layer to Network Layer

$$BW_{NETWORK} = BW_{TRANSPORT} \text{ (datagrams/s)}$$

$$DS_{NETWORK} = DS_{TRANSPORT} + HS_{NETWORK} \text{ (bytes/datagram),}$$

where $HS_{NETWORK}$ is the number of header bytes appended to the transport layer packet.

Suppose that “good” is selected as the user QoS requirement for the video service, and BW_{APPL} for the MPEG compression server is the same as the application layer bandwidth requirement provided in Table 6. The header lengths used for the transport and network layers are assumed to be 16 and 20 bytes, respectively. Table 9 captures the QoS translation process for Phase 1.

Table 9. QoS Translation from Network Layer to Application Layer

| LAYER | DATA SIZE | BANDWIDTH |
|-------------|-----------------------|-------------------|
| Application | 20,736 bytes/frame | 15–24 frames/s |
| Transport | 20,752 bytes/packet | 15–24 packets/s |
| Network | 20,772 bytes/datagram | 15–24 datagrams/s |

Assuming that the compression ratio is 8:1, Table 10 captures the QoS translation process for Phase 2.

Table 10. QoS Translation from Application Layer to Network Layer

| LAYER | DATA SIZE | BANDWIDTH |
|-------------|----------------------|-------------------|
| Application | 2,582 bytes/frame | 15–24 frames/s |
| Transport | 2,598 bytes/packet | 15–24 packets/s |
| Network | 2,618 bytes/datagram | 15–24 datagrams/s |

Delay. The total delay incurred in the compression server for a video frame consists of three parts: the delay associated with passing the uncompressed video frame from the network layer up to the application layer; the delay for queuing and MPEG compression; and the delay for passing the compressed video frame from the application layer down to the network layer. By applying the layer-by-layer QoS translation illustrated in Section 4, the total delay can be derived as follows:

Network Layer to Application Layer

$$\begin{aligned} D_{\text{TRANSPORT}\uparrow} &= D_{\text{NETWORK}\uparrow} + D_{\text{PROC}_{\text{TRANSPORT}\uparrow}}(s) \\ D_{\text{APPL}\uparrow} &= D_{\text{TRANSPORT}\uparrow} + D_{\text{PROC}_{\text{APPL}\uparrow}}(s), \end{aligned}$$

where D_{PROC} represents the delay associated with the layer and D is the accumulated delay associated with each layer.

Thus, the delay for the first part can be calculated as follows:

$$D_{\text{NETWORK-APPL}} = D_{\text{APPL}\uparrow} - D_{\text{NETWORK}\uparrow} + D_{\text{PROC}_{\text{NETWORK}\uparrow}}(s)$$

Application Layer. The second part of the delay can be represented as follows:

$$D_{\text{QUEUE-COMP}} = D_{\text{QUEUE}} + D_{\text{COMP}}(s),$$

where D_{QUEUE} and D_{COMP} represent queuing delay and MPEG compression delay, respectively.

Application Layer to Network Layer

$$\begin{aligned} D_{\text{TRANSPORT}\downarrow} &= D_{\text{APPL}\downarrow} + D_{\text{PROC}_{\text{TRANSPORT}\downarrow}}(s) \\ D_{\text{NETWORK}\downarrow} &= D_{\text{TRANSPORT}\downarrow} + D_{\text{PROC}_{\text{NETWORK}\downarrow}}(s). \end{aligned}$$

Therefore, the delay for the third part is the same as $D_{\text{NETWORK}\downarrow}$, illustrated as:

$$D_{\text{APPL-NETWORK}} = D_{\text{NETWORK}\downarrow}(s)$$

Finally, the total delay can be represented as follows:

$$D_{\text{TOTAL}} = D_{\text{NETWORK-APPL}} + D_{\text{QUEUE-COMP}} + D_{\text{APPL-NETWORK}}(s).$$

Assuming that the processing time incurred at each layer is a constant value of 1 ms, and the delay bound associated with MPEG compression is 5 ms, which is the sum of the time it takes to perform compression and the queuing delay, the total delay can be calculated as follows:

$$D_{\text{TOTAL}} = 3 * 1 + 5 + 3 * 1 = 11 \text{ (ms)}.$$

6.4.1.2 Resource Mapping

The processing resources of concern for the MPEG compression PRoS are the processing time incurred at each layer of the protocol stack for data manipulation and the processing time associated with the MPEG compression. Assuming that P is the processing capability of a CPU in instructions/s, and that N is the number of instructions required to perform the specific functions associated with each layer, then the amount of CPU processing time required at each layer, denoted by Q_{\uparrow} or Q_{\downarrow} , (where the up and down arrows represent receiving and sending direction, respectively) can be calculated as follows:

$$Q_{\uparrow} = Q_{\downarrow} = N/P \text{ (s)}$$

Thus, the total processing time required by the MPEG compression PRoS, represented as Q_{total} , can be calculated as follows:

$$Q_{\text{total}} = \sum_i Q_{\uparrow} + \sum_i Q_{\downarrow} + Q_{\text{COMP}}(s).$$

Here, i is the index number of each protocol layer wherein processing time is required, and Q_{COMP} represents the processing time for MPEG compression.

The storage resources of interest for the MPEG compression PRoS are the buffer space required at the network layer to absorb jitter for the incoming traffic and to provide loss-free guarantees for the incoming and outgoing traffic. By applying the RCSPQ* service discipline described in Appendix C, assuming the delay bounds of the schedulers for the incoming and outgoing traffic are dm_{in} and dm_{out} respectively, the buffer requirements described above are illustrated by Zhang and Ferrari [1992] as follows:

$$B_{incoming} = PDU_{max} * \left\lceil \frac{2 \times dm_{in} + dm_{prev} + dl_{prev}}{x_{min}} \right\rceil \text{ (bytes) .}$$

In the equation above, $B_{incoming}$ is the buffer space required at the network layer for the incoming traffic in bytes, PDU_{max} is the maximum size of a network layer datagram in bytes, X_{min} is the minimum datagram interarrival time, dm_{in} is the delay bound assigned at the scheduler for the incoming traffic, dm_{prev} is the delay bound assigned at the scheduler of the previous node, and dl_{prev} is the delay bound assigned at the communication subnetwork between the previous node and the compression server.

$$B_{outgoing} = PDU_{max} * \left\lceil \frac{dm_{out}}{x_{min}} \right\rceil \text{ (bytes) .}$$

In the equation above, $B_{outgoing}$ is the buffer space required at the network layer for the outgoing traffic in bytes, PDU_{max} is the maximum size of a network layer datagram in bytes, X_{min} is the minimum datagram interarrival time, and dm_{out} is the delay bound assigned at the scheduler for the outgoing traffic.

Assume that dm_{prev} , dm_{in} and dm_{out} are 50 ms each, dm_{prev} is 100 ms, and X_{min} is 1/15 s. For the incoming traffic, PDU_{max} represents the maximum size of an incoming datagram, which is the same as the network-layer data size in Table 9. For the outgoing traffic, PDU_{max} represents the maximum size of an outgoing datagram, which is the same as the network layer data size in Table 10. Thus, the above equations yield 83,088 and 2,618 bytes, respectively, for $B_{incoming}$ and $B_{outgoing}$.

6.4.2 Horizontal QoS Mapping for I/O LRoS

In this subsection, we show how horizontal mapping can be applied to an I/O LRoS. The I/O LRoS of interest transmits video data from the source machine to the MPEG compression server over an ATM network, as depicted in Figure 26.

6.4.2.1 QoS Translation

Horizontal QoS mapping translates the communication-protocol-independent QoS parameters, generated at the network layer of end-system nodes into parameters that are communication-protocol dependent for all the subnetworks along the connection path. These network-layer QoS parameters (bandwidth, loss ratio, error ratio, delay, and jitter) can be used to

*RCSPQ: Rate-Controlled, Static-Priority Queueing.

calculate the end-to-end QoS requirements for the underlying network represented as BW_{ete} , LR_{ete} , ER_{ete} , D_{ete} , and J_{ete} , respectively. Subsequently, these end-to-end QoS requirements are translated into subnetwork parameters that are protocol specific. The end-to-end bandwidth, loss ratio and error ratio requirements are the same as the corresponding network-layer QoS parameters, while the end-to-end delay QoS requirement for the underlying network can be calculated as follows:

$$D_{ete} = D_{total} - D_{NETWORK\downarrow} - D_{APPL\uparrow},$$

where D_{total} represents the ete delay for the application, $D_{NETWORK\downarrow}$ represents the accumulated delay from the application layer to the network layer, and $D_{APPL\uparrow}$ represents the accumulated delay from the network layer to the application layer.

In the following subsections, we will describe a generic QoS translation methodology that can be applied to any I/O LRoS, and provide a numerical example of QoS translation across ATM subnetworks.

6.4.2.1.1 Generic Translation Formulas

Bandwidth. Let us assume that there are N communication subnetworks along an end-to-end connection. In order to translate the end-to-end bandwidth requirement of the network layer at end-system nodes to the bandwidth parameter of each subnetwork along the path, the following condition must be satisfied:

$$\text{Min} \{ BW_i \} \geq (BW_{NETWORK} ; 1 \leq i \leq N ,$$

where BW_i represents the bandwidth requirement at the i th communication subnetwork.

The above condition states that the minimum bandwidth provided by any communication subnetwork along the path should be greater than or equal to the end-to-end bandwidth required at the network layer of end-system nodes ($BW_{NETWORK}$).

Delay. The end-to-end delay bound is calculated based on the following assumptions: (1) the end-to-end connection path between any pair of end-system nodes consists of communication subnetworks which are interconnected by routers or gateways, and (2) the communication subnetworks have bounded delays. Thus, the end-to-end delay bound between a pair of end nodes is considered as the sum of the delay bounds for datagram waiting time in the nodes and the delay bounds for the communication subnetworks:

$$D_{ete} = \sum_{i=1}^{N-1} (dm_i + dl_i) + dm_N ,$$

where N is the total number of nodes along the connection path including both end nodes, dl_i is the delay bound for i th subnetwork along the path, dm_i is the delay bound for a datagram with priority m_i waiting in the scheduler of node i , and dm_N is the delay bound for a datagram with priority m_N waiting in the scheduler of receiving end node N . The formula for deriving dm_i is provided as Theorem 1 by Zhang and Ferrari [1992].

Loss Ratio. Assuming that datagram loss is independent of each communication subnetwork along the ete connection and that each intermediate node on the path is loss free, then the end-to-end datagram loss ratio is 1 minus the product of the probabilities of successful delivery of a datagram. This can be expressed as follows:

$$LR_{ete} = 1 - \prod_i (1 - LR_i) ,$$

where LR_{ete} is the end-to-end datagram loss ratio, and LR_i is the datagram loss ratio within the i th communication subnetwork.

Error Ratio. Similarly, the end-to-end datagram error ratio can be calculated as follows:

$$ER_{ete} = 1 - \prod_i (1 - ER_i) ,$$

where ER_{ete} is the end-to-end datagram error ratio, and ER_i is the datagram error ratio within the i th communication subnetwork.

Jitter. According to the distributed mechanism for controlling jitter described in Appendix A and the assumption made for end-to-end delay calculations, the end-to-end jitter can be computed as follows:

$$J_{ete} = dm_N .$$

The above formula implies that the end-to-end jitter is equal to the delay bound of the datagram waiting time in the scheduler of the receiving end-system node N.

6.4.2.1.2 QoS Translation Across ATM Subnetworks

In this subsection, a numerical example shows the results of applying the methodology of horizontal QoS translation to the network configuration illustrated in Figure 26. The methodology involved in this case will translate QoS parameters generated at the network layer of an end-system node into QoS parameters which are understood by the ATM subnetworks along the connection path. For example, the datagram rate at the network layer will be translated into a cell rate at the ATM layer in an ATM subnetwork. The protocol stack used for the ATM subnetworks, shown in Figure 26, consists of an AAL5 layer, an ATM layer and some arbitrary physical layer. The translation procedure starts from the AAL5 layer and is completed at the ATM layer. Table 11 shows the translation results for the I/O LRoS of interest. The details of the translation process are provided by Wang et al. [1997].

Table 11. Translation of Network-Layer QoS to AAL5-SAR QoS and to ATM QoS

| LAYER | DATA SIZE | BANDWIDTH | ERROR RATIO | LOSS RATIO |
|----------|-----------------------|----------------------|-------------|------------|
| Network | 20,772 bytes/datagram | 15–24 datagrams/s | 0.8070 | 0.8070 |
| AAL5-SAR | 48 bytes/PDU | 6,495–10,392 PDUs/s | 0.0038 | 0.000001 |
| ATM | 53 bytes/cell | 6,495–10,392 cells/s | 0.0042 | 0.000001 |

6.4.2.2 Resource Mapping

In this subsection, we provide the calculations to determine the buffer requirements at both intermediate nodes and receiving end-system nodes by means of the Rate-Controlled Static-Priority Queuing service discipline described in Appendix C. The longest time a datagram can stay in a node is the sum of (1) the longest time a datagram can wait in the regulator and (2) the delay bound for the datagram waiting time in the scheduler. The longest time a datagram can wait in the regulator of an intermediate node or a receiving end node can be represented as the sum of (1) the delay bound for the datagram waiting time in the scheduler of the preceding node and (2) the delay bound for the communication subnetwork between the preceding node and the current node. The buffer space required at each node along the connection's path (excluding the sending end node) is calculated by Zhang and Ferrari [1992] as follows:

$$B_i = \text{PDU}_{\max} * \left\lceil \frac{dm_i + dm_{i-1} + dl_{i-1}}{x_{\min}} \right\rceil \quad (\text{bytes}) .$$

In the equation above, B_i is the buffer space required at node i in bytes, PDU_{\max} is the maximum network layer datagram size in bytes, X_{\min} is the minimum datagram interarrival time, dm_i is the delay bound assigned at the scheduler of node i , dm_{i-1} is the delay bound assigned at the scheduler of node $i-1$, and dl_{i-1} is the delay bound assigned on the communication subnetwork between node $i-1$ and node i .

In order to calculate the buffer space at the router which provides the interconnectivity between the two ATM subnetworks, as shown in Figure 26, the following assumptions are made:

- The delay bound guaranteed by the scheduler at the router and the end nodes is 50 ms.
- The delay bound for each ATM subnetwork is 100 ms.
- The maximum network layer datagram size is listed in Table 11.

Therefore, the buffer space required at the router, represented as B_{router} , can be calculated as follows:

$$B_{\text{router}} = 20772 * \left\lceil \frac{50 + 50 + 100}{\frac{1}{15} \times 1000} \right\rceil = 62316 \text{ (bytes)} .$$

6.5 CONCLUSION

In this section, we have used a teleseminar application consisting of video, audio, and whiteboard services as a functional scenario to illustrate the concepts described in the previous sections. We have also provided a detailed example of the recursive decomposition of the video service into a sequence of PRoSs and I/O LRoSs. We have then used the results of the video service decomposition to model to show the relationship between the different subcomponents of the video service. In order to provide end-to-end application QoS guarantees, the vertical QoS mapping approach should be applied to the PRoSs that reside in the end-system nodes, whereas the horizontal QoS mapping approach should be applied to the I/O LRoSs that transmit data between PRoSs. The vertical and horizontal QoS mapping approaches together provide a means of mapping application QoS requirements into system resource requirements, which can be used by the resource managers to perform resource allocation. Finally, we have provided numerical examples of the compression PRoS and the I/O LRoS, to illustrate the vertical and horizontal QoS mapping approaches.

7 CONCLUSION

7.1 ACCOMPLISHMENTS

SRI International and Stanford Telecommunications, Inc., under contract from Rome Laboratory, developed a conceptual framework for adaptive, integrated enterprise management. Such an enterprise management system will control and allocate the resources of the next generation of C⁴I enterprise, in accordance with the users' QoS requirements, and will consider changes in operational modes, variations in the availability of resources, mobility, and anomalies. Such an adaptive, global resource management system will enable large, heterogeneous, distributed C⁴I systems to transition from peacetime operation to crisis response without any interruption of service; to utilize limited system resources effectively and efficiently; and to continue to operate with some level of functionality despite failures and anomalies.

This effort, known as the Global Resource Management project, began on 1 November 1995, and had an 18-month period of performance, under Contract F30602-95-C-0299.

This final technical report summarizes the work performed during the entire project, namely, the development of an adaptive, scalable architecture for QoS-driven resource management of distributed systems. Each section of the report describes components of this architecture. In the following pages, we summarize the previous sections of this report.

In Section 1, we analyze the need for resource management, particularly end-to-end resource management of distributed systems running diverse applications. We also identify the characteristics that a good resource management scheme must possess, including the following:

- Use a comprehensive definition of QoS
- Provide end-to-end QoS support over heterogeneous resources
- Provide integrated QoS support for hard-, soft-, and non-real-time applications
- Be adaptive to changes in system conditions.

We then identify previous work in this area. We conclude that although there had been extensive work in the both the network and distributed computing domains, none of the previous work possessed the required characteristics listed above. We then introduce our resource management architecture, as described in the remainder of the report.

In Section 2 we develop a taxonomy of QoS specifications, which is the starting point for defining the QoS interfaces between the different system components. The key to understanding QoS-based, distributed systems is to recognize the relationships between the different ways that QoS concepts are defined by different system components. The taxonomy that we present clearly identifies the different classes of QoS parameters. We see that the fundamental metrics are those of performance, cost, and security. The performance metrics include the classifications of timeliness, precision, and accuracy. The concept of these three primary classes of performance is important because most of the QoS specifications we have seen are just statements of one or two of these classes. Each class of the QoS specification includes requirements for absolute quantities and consistency measures. These requirements give a large, dynamic range of parameter specifications. Finally, the concept of a statistical distribution of each one of the parameters enables the

specification of parameter tolerance and variability. The classification of QoS metrics into their most basic groupings provides insights into the common mechanisms for translating QoS specifications between system layers.

In Section 3 we describe the models we have developed for capturing the application, resource, and system perspectives necessary for resource management. We first describe two innovative models to capture the application perspective. The first model, denoted the Logical Application Stream Model, recursively captures a distributed application's structure, resource requirements, and relevant end-to-end QoS parameters. When a user invokes the application, the resource manager can use the LASM to initially structure the end-to-end application, allocate resources to this application, and schedule this application on these resources, so as to provide QoS to all applications and to utilize system resources efficiently; later, when the system state changes, the resource manager can use this application model to reallocate, reschedule, and restructure applications dynamically. The recursive nature of the model enables application developers to easily model large-scale applications. We also describe a model, denoted the Benefit Function, that captures user QoS preferences and enables the resource manager to gracefully degrade application QoS under certain conditions.

In Section 4 we describe our model for describing distributed systems that supports distributed applications with QoS guarantees. This hierarchical system model allows the different types of systems with distributed, centralized, or hybrid control schemes to be modeled by means of a common set of constructs. The primary purpose of the model is to manage resources in an intelligent manner that compensates for changes in the system state and guarantees QoS to the applications being executed on the system.

The hierarchical structure of the system model complements our recursive application model. Information about the end-to-end application state and the state of the resources being used by the application can be considered the "first order" of information, since they directly affect the performance of the application from a user's perspective. The first-order information plays a critical role in adaptive resource management. Other, "second-order" information (such as the other applications in the system and the state of other resources that may affect the current application) is also needed by the managers. Second-order information affects the performance of the applications indirectly: e.g., a fault on a resource, not related to the application in consideration, may spill some of the load from that resource to the resource being used by the application under consideration, and therefore will cause the degradation of application QoS. The subsystem managers at the different levels of the subsystem hierarchy consider the global and local state information of the applications and the resources at different resolutions.

Another feature of the system and resource models is that they hide many implementation-specific details behind interfaces, so that heterogeneous resources can be modeled within the same system model. Resource-specific details are hidden inside the resource model, and the system interacts with the resources via the interfaces, which are resource independent. This approach also enables us to use uniform management mechanisms for heterogeneous systems.

In Section 6 we identify the set of algorithms that constitute the resource management scheme, including QoS translation, allocation, scheduling, trade-offs, and analysis. We also describe (1) the objective of the algorithms, (2) previous work in each area, (3) missing components that are necessary for end-to-end adaptive resource management, and (4) our approach

to solving these problems. In Section 5, we describe how the CORBA middleware infrastructure could be used to aid resource management. In Section 6, we conclude with an example illustrating the uses of resource management for a teleseminar application.

7.2 PUBLICATIONS AND PRESENTATIONS

During the course of this project, we wrote the following conference papers:

- Davis, M., and J. Sydir. 1996. "Resource Management for Complex Distributed Systems," *Proc. WORDS '96* (February).
- Sabata, B., S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence. 1997. "Taxonomy for QoS Specifications," *WORDS '97* (February).
- Sydir, J., B. Sabata, and S. Chatterjee. 1997. "End-to-End, Qos-Driven Resource Management for the Next-Generation Internet," accepted for presentation at the Next Generation Internet Workshop (12-14 May).
- Chatterjee, S., J. Sydir, B. Sabata, M. Davis, and T. Lawrence. 1997. "Modeling Distributed Applications with QoS Requirements," in *Proc. 2nd IEEE High Assurance Systems Engineering Workshop* (August).
- Chatterjee, S., B. Sabata, J. Sydir, C. Hammond, P. Clark, H. Hguyen, W. Wang, and T. Lawrence. 1997. "Global Resource Management," presented at the 3rd International Command and Control Research and Technology Symposium, Washington, D.C. (June).

We also wrote the following technical reports:

- Sabata, B., S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence. 1997. *Hierarchical Modeling of Systems for QoS Based Distributed Resource Management*, Technical Report, SRI International, Menlo Park, California.
- Wang, W., H. Hguyen, C. Hammond, and T. Lawrence. 1997. *An Approach to Mapping Multimedia Application QoS to Resources*, Technical Report, Stanford Telecommunications, Reston, Virginia.
- Sydir, J., S. Chatterjee, M. Davis, B. Sabata, and T. Lawrence. 1997. *Relationship between security and QoS driven resource management*, Technical Report, SRI International, Menlo Park, California.

We also gave oral presentations of our research results, as follows:

- 1995: Rome Laboratory C3A/B Technology Exchange Meeting
- 1996: Rome Laboratory C3A/B Technology Exchange Meeting
- 1996: Presentation at the Research, Development, Test and Evaluation Division (NRaD) of the Naval Command, Control, and Ocean Surveillance Center (NCCOSC), San Diego, California
- 1997: Rome Laboratory Final Report Presentation.

7.3 FUTURE WORK

The work presented in this report forms a framework on which we are developing an adaptive QoS-driven end-to-end resource management scheme. Our current focus is on developing the algorithms for adaptive resource management, and improving the models, if needed, to provide sufficient information to these algorithms. We will next implement these models and these algorithms, build a prototype distributed system, and run a set of representative applications to determine the value of our resource management scheme. Analysis on the testbed, as well as analysis using simulation, will be done to complete the work.

REFERENCES

- Anderson, D., R. Herrtwich, and C. Schaefer. 1990. *SRP: A Resource Reservation Protocol for Guaranteed Performance Communication in the Internet*, Technical Report TR-90-006, University of California at Berkeley (February).
- Barringer, B., T. Burd, F. Burghardt, A. Burstein, A. Chandrakasan, R. Doering, S. Narayanaswamy, T. Pering, B. Richards, T. Truman, J. Rabaey, and R. Brodersen. 1994. "Infopad: A System Design for Portable Multimedia Access," *Proc. Calgary Wireless 94 Conference* (July).
- Beck, J., and D. Siewiorek. 1994. *Automated Task Allocation and Processor Specification Strategies for Multi-computer Systems*, Technical Report CMU-EDRC 18-50-94, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Bettati, R., and J. Liu. 1990. "Algorithms for End-to-End Scheduling to Meet Deadlines," in *Proc. 2nd IEEE Conference on Parallel and Distributed Systems* (December).
- Campbell, A., C. Aurrecoechea, and L. Hauw. 1996. "A Review of QoS Architectures," in *Proc. 4th IFIP International Workshop on Quality of Service (IWQoS '96)*, Paris, France (March).
- Campbell, A., G. Coulson, F. Garcia, D. Hutchinson, and H. Leopola. 1993. "Integrated Quality of Service for Multimedia Communications," *Proc. IEEE Infocom '93*, San Francisco, California, pp. 732-739 (March).
- Campbell, A., G. Coulson, and D. Hutchison. 1994. "A Quality of Service Architecture," *ACM SIGCOMM, Computer Communication Review*, Vol. 24, pp. 6-27 (April).
- Campbell, A., G. Coulson, and D. Hutchison. 1997. "Supporting Adaptive Flows in a Quality of Service Architecture," *ACM Multimedia Systems Journal* (to appear).
- Campbell, A., C. Aurrecoechea, and L. Hauw. 1996. "A Review of QoS Architectures," *Proc. 4th IFIP International Workshop on Quality of Service (IWQS '96)*, Paris, France (March).
- Chatterjee, S. 1997. "QUASAR, An Adaptive Quality of Service Architecture for Resource Management of Distributed Real Time Systems," <http://www.erg.sri.com/projects/quasar> (June).
- Chatterjee, S. 1997. "A Quality of Service-Based Allocation and Routing Algorithm for Distributed, Heterogeneous Real Time Systems," in *17th IEEE International Conference on Distributed Computing Systems* (May).
- Chatterjee, S., M. Davis, B. Sabata, J. Sydir, and T. Lawrence. 1997. *Modeling Distributed Applications with QoS Requirements*, ITAD Technical Report, SRI International, Menlo Park, California (in preparation).
- Chatterjee, S., B. Sabata, J. Sydir, and T. Lawrence. 1997. *Benefit Functions for QoS Trade-Offs*, ITAD Technical Report, SRI International, Menlo Park, California (in preparation).
- Chatterjee, S., and J. Strosnider. 1995a. "Distributed Pipeline Scheduling: End-to-End Analysis of Heterogeneous, Multi-Resource Real-Time Systems," *Proc. 15th International Conference on Distributed Computing Systems*, Vancouver, Canada (May).

- Chatterjee, S., and J. Strosnider. 1995b. "A Generalized Admissions Control Strategy for Heterogeneous, Distributed Multimedia Systems," in *Proc. ACM Multimedia '95*, San Francisco, California (November).
- Chatterjee, S., and J. Strosnider. 1995c. "Distributed Pipeline Scheduling: A Framework for Distributed, Heterogeneous Real-Time System Design," in *The Computer Journal (British Computer Society)*, Vol. 38, No. 4.
- Chatterjee, S., and J. Strosnider. 1996. "Quantitative Analysis of Hardware Support for Real-Time Operating Systems," *Journal of Real-Time Systems* 10, pp. 123–142.
- Chatterjee, S., and J. Strosnider. 1997. "A Quality of Service-Based Allocation and Routing Algorithm for Distributed, Heterogeneous Real Time Systems," in *Proc. 17th IEEE International Conference on Distributed Computing Systems (ICDCS97)*, Baltimore, Maryland (May).
- Chatterjee, S., J. Sydir, B. Sabata, M. Davis, and T. Lawrence. 1997. "Modeling Distributed Applications with QoS Requirements," in *Proc. 2nd IEEE High Assurance Systems Engineering Workshop* (August).
- Chatterjee, S., J. Sydir, B. Sabata, and T. Lawrence. 1997. "Modeling Applications for Adaptive QoS-based Resource Management," to be presented at HASE '97, Bethesda, Maryland (August).
- Chatterjee, S., B. Sabata, J. Sydir, C. Hammond, P. Clark, H. Hguyen, W. Wang, and T. Lawrence. 1997. "Global Resource Management," presented at the 3rd International Command and Control Research and Technology Symposium (June).
- Clark, D., S. Shenker, and L. Zhang. 1992. "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," in *Proc. SIGCOMM '92*, pp. 14–26 (August).
- Colquist, J. 1996. *Real Time System Analysis Using the Burdened Task Model*, M.S. Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania (December).
- Coolahan, J., and N. Roussopoulos. 1983. "Timing Requirements for Time-Driven Systems Using Augmented Petri-Nets," *Proc. IEEE Trans. Software Engineering*, pp. 603–616 (September).
- Daigle, S., and J. Strosnider. 1994. "Disk Scheduling of Continuous Media Data Streams," *SPIE Conference on High-Speed Networking and Multimedia Computing*.
- Davis, M., and J. Sydir. 1996. "Resource Management for Complex Distributed Systems," *WORDS '96* (February).
- Davis, M., A. Downing, and T. Lawrence. 1994. "Adaptable Resource Management for Soft Real-Time Systems," presented at the Symposium on Command and Control Research and Decision Aids, Monterey, California (June).
- Ferrari, D., and D. Verma. 1990. "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal of Selected Areas of Communications*, 8(3), pp. 368–379 (April).
- Ferrari, D., J. Ramaekers, and G. Ventre. 1992. "Client-Network Interactions in Quality of Service Communications Environments," available on the Internet at <ftp://tenet.berkeley.edu/pub/tenet/Papers/FeRaVe92.ps>.
- Gilles, D., and J. Liu. 1991. *Scheduling Tasks with AND/OR Precedence Constraints*, Technical Report UIUCDCS-R-90-1627, University of Illinois, Urbana-Champaign (20 March).

- Harbour, M., M. Klein, and J. Lehoczky. 1991. "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority," in *Proc. 1991 Real-Time Systems Symposium* (December).
- Hull, D., and J.W.S. Liu. 1993. "ICS: A System for Imprecise Computations," *Proc. AIAA Computing in Aerospace 9*, San Diego, California, pp. 371-374 (October).
- Jeffay, K. 1993. "The Real-Time Producer/Consumer Paradigm: A Paradigm for the Construction of Efficient, Predictable Real-Time Systems," in *ACM/SIGAPP Symposium on Applied Computing* (February).
- Jeffay, K., and D. Stone. 1993. "Accounting for interrupt handling costs in dynamic priority task systems," in *Proc. 1993 IEEE Real-Time Systems Symposium*.
- Katcher, D., H. Arakawa, and J. Strosnider. 1992. "Engineering and Analysis of Real-Time Microkernels," *Proc. 9th IEEE Workshop on Real-Time Operating Systems and Software*, 1(1), pp. 15-19 (May).
- Katcher, D., H. Arakawa, and J. Strosnider. 1993. "Engineering and Analysis of Fixed Priority Schedulers," in *IEEE Trans. Software Engineering*, 19(9) (September).
- Katcher, D., H. Arakawa, and J. Strosnider. 1994. "Modeling DSP Operating Systems for Multimedia Applications," in *Proc. 1994 Real-Time Systems Symposium* (April).
- Kettler, K., and J. Strosnider. 1994. "Scheduling Analysis of the Micro Channel Architecture for Multimedia Applications," in *Proc. IEEE International Conference on Multimedia Computing and Systems* (May).
- Kettler, K., J. Lehoczky, and J. Strosnider. 1995. "Modeling bus scheduling policies in real-time systems," in *Proc. 16th IEEE Real-Time Systems Symposium*.
- Lee, Y., and L. Shen. 1990. "Real time communication in multiple token ring networks," *Proc. 11th IEEE Real-Time Systems Symposium*, pp. 146-153 (December).
- Maffeis, S., and D. Schmidt. 1997. "Constructing Reliable Distributed Communication Systems with CORBA," *IEEE Communications Magazine*, Vol. 35, No. 2, pp. 56-60 (February).
- Mercer, C., S. Savage, and H. Tokuda. 1993. "Processor Capacity Reserves: An Abstraction for Managing Processor Usage," in *Proc. Fourth Workshop on Workstation Operating Systems*, Napa, California (October).
- Nahrstedt, K., and J. Smith. 1993. "An Application-Driven Approach to Networked Multimedia Systems," *Proc. 18th Conference on Local Computer Networks*, Minneapolis, Minnesota, pp. 361-368 (September).
- Nahrstedt, K., and J. Smith. 1995. "The QoS Broker," *IEEE Multimedia*, Spring, pp. 53-67.
- Nahrstedt, K., and R. Steinmetz. 1995. "Resource Management in Networked Multimedia Systems," *IEEE Computer*, pp. 52-63 (May).
- Nanda, A., and D. Stenger. 1991. *Simulated annealing algorithms for scheduling directed task graphs on multiprocessors*, Technical Report TAMU 91-040, Computer Science Department, Texas A&M University, Galveston, Texas.
- OMG. 1995. *The Common Object Request Broker: Architecture and Specification*, Object Management Group, Inc. and X/Open Co. Ltd., Revision 2.0 (July).

- Parris, C., and D. Ferrari. 1993. *A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet Switching Integrated Services Networks*, Technical Report TR-93-005, University of California at Berkeley.
- Parris, C., G. Ventre, and H. Zhang. 1993. "Graceful Adaptation of Guaranteed Performance Service Connections," in *Proc. Globecom '93*, Houston, Texas (November).
- Perlman, R. 1992. *Interconnections: Bridges and Routers*, Addison-Wesley.
- Raha, A. 1995. *Real-Time Communication in ATM Networks*, Ph.D. Thesis, Department of Computer Science, Texas A&M University, Galveston, Texas.
- Ramamritham, K. 1989. *Allocation and Scheduling of Complex Periodic Tasks*, Technical Report 90-01, University of Massachusetts, Amherst (January).
- Rampal, S., D. Reeves, and D. Agrawal. 1994. "An Evaluation of Routing and Admission Control Algorithms for Multimedia Traffic in Packet-switched Networks," *Proc. 5th IFIP Conference on High-Performance Networking*, pp. 77-92 (June).
- Sabata, B., S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence. 1997. "Taxonomy for QoS Specifications," in *Proc. IEEE Computer Society 3rd International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '97)*, Newport Beach, California (February).
- Sabata, B., S. Chatterjee, J. Sydir, and T. Lawrence. 1997. *Hierarchical Modeling of Systems for QoS-Based Distributed Resource Management*, ITAD Technical Report, SRI International, Menlo Park, California (in preparation).
- Sathaye, S. 1993. *Scheduling Real-Time Traffic in Packet Switched Networks*, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Sathaye, S., and J. Strosnider. 1992. *Conventional and early token release scheduling models of the IEEE 802.5 token ring*, Technical Report CMU CDS-92-9, Carnegie Mellon University, Pittsburgh, Pennsylvania (September).
- Schmidt, D., A. Gokhale, T. Harrison, and G. Paruckar. 1997. "A High-Performance End System Architecture for Real-Time CORBA," *IEEE Communications Magazine*, Vol. 35, No. 2, pp. 72-77 (February).
- Stewart, D., D. Schmitz, and P. Khosla. 1989. "Chimera II: A Real-Time Multiprocessing Environment for Sensor-Based Control," in *Proc. IEEE International Symposium on Intelligent Control*.
- Strosnider, J., and T. Marchok. 1989. "Responsive, deterministic IEEE 802.5 token ring scheduling," *Journal of Real-Time Systems 1*, pp. 133-158.
- Sydir, J., S. Chatterjee, M. Davis, B. Sabata, and T. Lawrence. 1997a. *Relationship Between Security and QoS-Driven Resource Management*, ITAD Technical Report, SRI International, Menlo Park, California (in preparation).
- Sydir, J., S. Chatterjee, B. Sabata, and T. Lawrence. 1997b. *QUASAR: Quality of Service Architecture for Resource Management*, ITAD Technical Report, SRI International, Menlo Park, California (in preparation).
- Sydir, J., B. Sabata, and S. Chatterjee. 1997. "End-to-End, Qos-Driven Resource Management for the Next-Generation Internet," accepted for presentation at the Next Generation Internet Workshop (12-14 May).

- Tindell, K., A. Burns, and A. Wellings. 1992. "Allocating hard real-time tasks: an NP-hard problem made easy," *Real-Time Systems Journal*, Vol. 4, No. 2, pp. 145–165 (June).
- Verma, D., H. Zhang, and D. Ferrari. 1991. "Guaranteeing Delay Jitter Bounds in Packet-Switched Networks," *Proc. Tricomm 91*, pp. 35–46 (April).
- Vogel, A., B. Kerherve, G. Bochmann, and J. Gecsei. 1995. "Distributed Multimedia and QOS: A Survey," *IEEE Multimedia*, Summer, pp. 10–19.
- Vogt, C., R. Herrtwich, and R. Nagarajan. 1992. *HeiRAT—The Heidelberg Resource and Administration Technique: Design Philosophy and Goals*, IBM Technical Report 43.9213, presented at Kommunikation in Verteilen Systemen, Munich (March).
- Wang, W., H. Hguyen, C. Hammond, and T. Lawrence. 1997. *An Approach to Mapping Multimedia Application QoS to Resources*, Technical Report, Stanford Telecommunications, Sunnyvale, California.
- Wolfe, V., and R. Johnston. 1997. "Real-Time Technologies RFI Response: Real-Time CORBA in the DHDA Project," Real-Time Object-Oriented Systems Group, Department of Computer Science, University of Rhode Island, Kingston, available at http://www.cs.uri.edu/rtisorac/pubs/RFI_RESP.html.
- Zinky, J., D. Bakken, and R. Schantz. 1997a. *Supporting Quality of Service for CORBA Objects*, Report No. 8119, Bolt Beranek and Newman Inc. Systems and Technologies, Cambridge, Massachusetts.
- Zinky, J. D. Bakken, and R. Schantz. 1997b. "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems (Special Issue on the OMG and CORBA)*, John Wiley and Sons, Inc. (January).
- Zhang, H., and D. Ferrari. 1992. "Rate-Controlled Static Priority Queuing," *Proc. Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, California (November).

Appendix A

OOA RESOURCE MODEL OBJECT SPECIFICATION

OOA RESOURCE MODEL OBJECT SPECIFICATION

Section 3 of the main text described the resource and system models used for the GRM project. This appendix presents an alternative object-oriented methodology for modeling resources. Future work may combine the two models into a single model. Object-oriented analysis (OOA) techniques were selected for use in developing the OOA Resource Model, since OOA provides a well-structured, implementation independent means of decomposing a system's objectives and identifying the roles and relationships among its resource components. In addition, the use of OOA techniques (e.g., the use of C++) provides a smooth transition to a future object-oriented design phase. The OOA Resource Model, also referred to as the Resource Information Model, characterizes resources in terms of objects and attributes. The pertinent associations between the resource entities are abstracted as relationships between objects.

In developing the GRM Resource Model, we followed the methods and notation defined by Shlaer and Mellor,* where objects are simply abstractions of any real-world entity. The characteristics of an object are defined as attributes that must be possessed by each instance of an object. Key attributes serve to uniquely distinguish each instance of an object and are defined as identifiers for an object. Object associations are abstracted as relationships and systematically hold between objects. Relationships are characterized as conditional and unconditional and may exist in one-to-one, one-to-many, or many-to-many ties between objects. The reader is referred to Shlaer and Mellor [ibid.] for further information related to the construction of and notation to represent, objects and their relationships.

As shown in Figure A-1, the Resource Model is a hierarchical structure classifying like objects (resources) according to the types of attributes that define each resource's QoS capabilities in terms of timeliness, precision, and accuracy. This classification method is necessary because different resource classes have different means of specifying QoS parameters. For example, a measure of the timeliness of a processor is processing speed, while a measure of timeliness for a storage device is access time.

The first level of decomposition classifies all resources as processing, storage, or communication resources, depicted as objects 2, 3, and 4 in Figure A-1. Each of these has attributes that are unique to that abstraction and pertain to the QoS parameters of that resource. The processing resource object is further decomposed into a CPU (object 5) or a specialized processor, such as a graphic processor or a mathematics coprocessor (object 6). Similarly, the communication resource is decomposed into adapter type communication resources for digital transmission (object 7) and modem communication resources for analog transmission (object 8). Adapter communication resources further decompose into switches and routers (objects 9 and 10).

The remainder of the resource model is devoted to an abstraction of the interconnections of communication resources. A relationship is modeled between the communication resource (object 4) and the communications protocol (object 11), to account for the effects that the choice of protocols, such as network protocol, routing algorithm, or network management protocol, have on the QoS of the transmission. An example of these effects would be the difference in latency

*Shlaer, S., and S. Mellor. 1988. *Object-Oriented Systems Analysis: Modeling the World in Data*, Prentice Hall, Englewood Cliffs, New Jersey.

Shlaer, W. and S. Mellor. 1992. *Object-Oriented Systems Analysis: Modeling the World in States*, Prentice Hall, Englewood Cliffs, New Jersey.

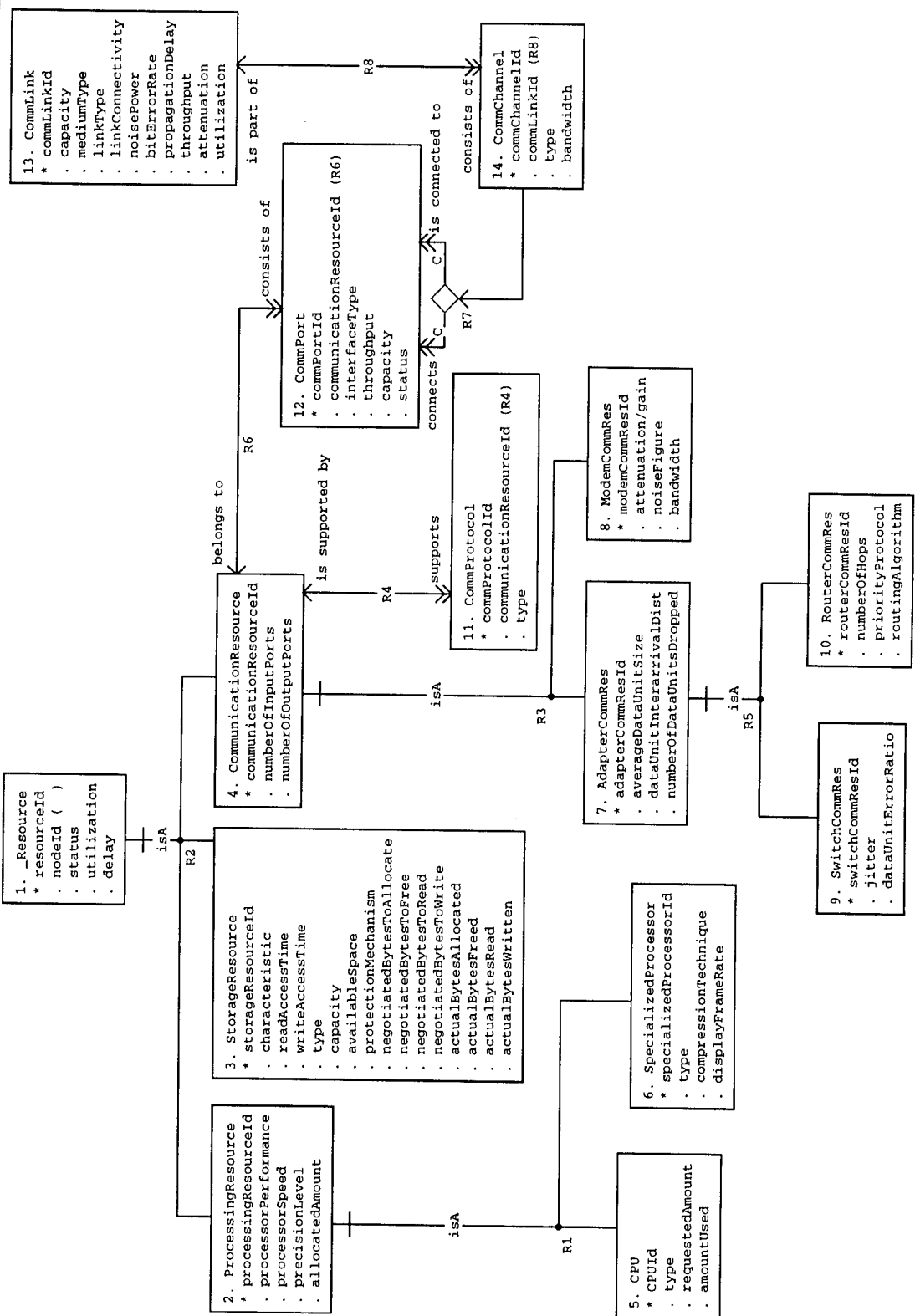


Figure A-1. Object-Oriented Resource Model

(timeliness) between transmissions using go-back-N or selective ACK automatic repeat request (ARQ) types. The communications channel (object 14) is modeled as the association between (i.e., the connection of) two or more communications devices through their respective communication ports (object 12). The physical characteristics of the communication link (object 13), such as a fiber-optic cable or satellite link, define many of the QoS capabilities of the channel.

It is important to note that the instantiations of these abstract objects do not necessarily map exactly to a hardware device. Many actual hardware devices support two or three of the resource functions modeled, i.e., processing, storage, and communications functions. The individual functions of the resource have been decomposed in the modeling process in order to more effectively group QoS parameters.

As described by Shlaer and Mellor, Object and Relationship Specifications further define the objects themselves, their attributes, and the relationships between objects. The remainder of this appendix describes the Object Specification.

The Resource Model Object Specification (RMOS) provides a textual description of the Resource Model.* The RMOS numerically lists the name of each object as it appears in the Resource Model. With each enumerated object, attributes are listed with the key attribute or identifier indicated by an asterick (*). Textual descriptions are provided for each object and its associated attributes. The Domain description for an attribute indicates whether the data can be represented as an identification number (Id), integer value, real number, or character string. The relationships between the objects contained in the RMOS are described further in the Resource Model Relationship Specification (RMRS) contained in Appendix B. Used together, the RMOS and RMRS provide a formal textual description of the entire Resource Model.

1. _Resource ()

(resourceId,
nodeId,
status,
utilization,
delay)

Identifiers:

* [resourceId]

Description:

This object describes a hardware resource within a node that is of interest to QoS management. The resource can be a processing resource, a storage resource, or a communication resource. The object represents the resource hardware that is allocated to a unit of work (UoW) to perform a particular thread.

1.1 _Resource.resourceId

Description:

This attribute uniquely identifies the resource.

*The Resource Model is shown in Figure 4, Section 3 of the main text of this report.

Domain:

Id

1.2 _Resource.nodeId

Description:

This attribute formalizes the relationship “Node contains Resource.”

Domain:

Id

1.3 _Resource.status

Description:

This attribute specifies the status of the resource. Its domain includes

1. Off-line
2. On-line
3. Busy.

Domain:

String

1.4 _Resource.utilization

Description:

This attribute specifies the utilization of a resource. Examples are:

1. The time a processor is in use divided by a given period of time
2. The used space divided by the capacity of a storage resource
3. The ratio of the number of data units forwarded by communication resource and the number of data units entering the resource.

Domain:

Real

1.5 _Resource.delay

Description:

This attribute specifies the processing delay of a resource.

Domain:

Real

2. ProcessingResource ()

(processingResourceId,
processorPerformance,
processorSpeed,
precisionLevel,

allocatedAmount)

Identifiers:

* [processingResourceId]

Description:

This object represents the processing resource within a node. The processing resource consists mainly of a central processing unit (CPU) that performs the primary computing operations for the node.

The processing resource can also include specialized processing units that provide additional computing power and aid the CPU in performing specific tasks. Examples of specialized processors include an I/O controller, a graphic processor, and a math co-processor.

Attributes of interest for the ProcessingResource object include performance in terms of the number of instructions executed per second in the case of a CPU, or the number of floating point operations per second in the case of a math co-processor; the speed of the processor in kHz; and precision level as the number of decimal places in a floating point computation result or the number of bits per video or audio frame.

2.1 ProcessingResource.processingResourceId

Description:

This attribute uniquely identifies the processing performed for a given process.

Domain:

String

2.2 ProcessingResource.processorPerformance

Description:

This attribute provides the performance of the supporting processor in terms of the number of instructions per second, the number of floating point operations per second, or the number of frames processed per second.

Domain:

Integer

2.3 ProcessingResource.processorSpeed

Description:

This attribute specifies the clock rate supported by a processor in kHz.

Domain:

Real

2.4 ProcessingResource.precisionLevel

Description:

This attribute specifies the precision supported by a particular processor for a process. If it is a computation, then the attribute specifies the number of decimal places for the result; if it is video processing, then the attribute specifies the ratio for a particular compression technique.

Domain:

Integer

2.5 ProcessingResource.allocatedAmount

Description:

This attribute specifies the time allocated to a UoW.

Domain:

Time

3. StorageResource ()

(storageResourceId,
characteristic,
readAccessTime,
writeAccessTime,
type,
capacity,
availableSpace,
protectionMechanism,
negotiatedBytesToAllocate,
negotiatedBytesToFree,
negotiatedBytesToRead,
negotiatedBytesToWrite,
actualBytesAllocated,
actualBytesFreed,
actualBytesRead,
actualBytesWritten)

Identifiers:

* [storageResourceId]

Description:

This object describes the storage resource that is allocated to a UoW. The storage resource can be further divided into two types:

1. Volatile memory such as RAM or caches used to store application codes in execution
2. Nonvolatile memory such as disks, tapes, and hard drives used for permanent storage.

Common attributes for the storage resource include the read and write access time to a memory device, the device type, and the device capacity.

3.1 StorageResource.storageResourceId

Description:

This attribute uniquely identifies the storage allocated to a process specified by capsuleId.

Domain:

Id

3.2 StorageResource.characteristic

Description:

This attribute specifies the volatile or nonvolatile characteristic of the storage.

Domain:

String

3.3 StorageResource.readAccessTime

Description:

This attribute specifies the read access time for storage, in seconds.

Domain:

Real

3.4 StorageResource.writeAccessTime

Description:

This attribute specifies the write access time for a storage device. If the write access time is infinite, that implies that the device is read-only capable.

Domain:

Real

3.5 StorageResource.type

Description:

This attribute specifies the storage device type. Its domain includes

1. RAM
2. Hard drive
3. Bernoulli disk
4. CD-ROM
5. Cache
6. Laser disc
7. Diskette
8. Floppy disk
9. Worm
10. Magneto-optic
11. Floptical

- 12. ROM
- 13. EEPROM
- 14. PROM
- 15. Other.

Domain:
String

3.6 StorageResource.capacity

Description:
This attribute specifies the storage capacity in bytes.

Domain:
Integer

3.7 StorageResource.availableSpace

Description:
This attribute specifies the unused space in a StorageResource. This could be in terms of bytes, pages, or blocks.

Domain:
Integer

3.8 StorageResource.protectionMechanism

Description:
This attribute specifies the memory protection scheme used.

Domain:
String

3.9 StorageResource.negotiatedBytesToAllocate

Description:
This attribute specifies the number of bytes to be allocated as negotiated during QoS negotiation.

Domain:
Integer

3.10 StorageResource.negotiatedBytesToFree

Description:
This attribute describes the number of bytes to be made available as negotiated.

Domain:
Integer

3.11 StorageResource.negotiatedBytesToRead

Description:

This attribute specifies the number of bytes to be read as negotiated.

Domain:

Integer

3.12 StorageResource.negotiatedBytesToWrite

Description:

This attribute specifies the number of bytes to be written as negotiated.

Domain:

Integer

3.13 StorageResource.actualBytesAllocated

Description:

This attribute describes the number of bytes actually allocated.

Domain:

Integer

3.14 StorageResource.actualBytesFreed

Description:

This attribute describes the number of bytes actually freed.

Domain:

Integer

3.15 StorageResource.actualBytesRead

Description:

This attribute describes the number of bytes actually read.

Domain:

Integer

3.16 StorageResource.actualBytesWritten

Description:

This attribute describes the number of bytes actually written.

Domain:

Integer

4. CommunicationResource ()

(communicationResourceId,
numberOfInputPorts,
numberOfOutputPorts)

Identifiers:

* [communicationResourceId]

Description:

This object describes the communication hardware resource that is necessary to provide a node with access to the communications channel. Examples of such resources are communications adapters (or network interface cards) for digital transmission, and modems (for analog transmission). Both of these types of resources are also known as data circuit-terminating equipment (DCE).

Attributes associated with this type of object used in digital transmission include the number of input and output ports, the average size of a data unit (packet, cell, datagram, etc.), and the distribution characteristic of incoming data units.

Attributes for the object used in analog transmission include the noise figure of the device, the bandwidth supported, and the range of transmission covered by the device.

4.1 CommunicationResource.communicationResourceId

Description:

This attribute uniquely identifies the communication hardware resource that provides the necessary network interface for a node. The hardware could be a LAN adapter supporting various protocols such as Ethernet, token ring, and ATM.

Domain:

Id

4.2 CommunicationResource.numberOfInputPorts

Description:

This attribute specifies the number of input ports supported by the CommunicationResource hardware.

Domain:

Integer

4.3 CommunicationResource.numberOfOutputPorts

Description:

This attribute specifies the number of output ports supported by the CommunicationResource hardware.

Domain:
Integer

5. CPU ()

(CPUId,
type,
requestedAmount,
amountUsed)

Identifiers:
* [CPUId]

Description:
This object represents the CPU that provides the primary computing capability for a node. Attributes of the CPU resource object include the CPU type and the requested, allocated, and used amounts of resource in terms of time units associated with a particular UoW.

5.1 CPU.CPUId

Description:
This attribute uniquely identifies the CPU object.

Domain:
Id

5.2 CPU.type

Description:
This attribute specifies the type of CPU (Pentium, 80486, 68000 . . .).

Domain:
String

5.3 CPU.requestedAmount

Description:
This attribute describes the amount of time requested by the UoW.

Domain:
Time

5.4 CPU.amountUsed

Description:
This attribute describes the amount of the resource (in time units) actually used by the UoW.

Domain:
Time

6. SpecializedProcessor ()

(specializedProcessorId,
type,
compressionTechnique,
displayFrameRate)

Identifiers:

* [specializedProcessorId]

Description:

This object represents a processor unit with specialized processing capability to assist the CPU in handling specific tasks and to speed up the execution of an application thread (UoW) requiring special handling, such as the transfer of large data files to and from local storage. Examples of this object include a graphic processor, an I/O processor, and a math coprocessor.

6.1 SpecializedProcessor.specializedProcessorId

Description:

This attribute uniquely identifies the SpecializedProcessor object.

Domain:

Id

6.2 SpecializedProcessor.type

Description:

This attribute specifies the type of specialized processor.

Domain:

String

6.3 SpecializedProcessor.compressionTechnique

Description:

This attribute specifies the compression techniques employed by the specialized processor.

Domain:

String

6.4 SpecializedProcessor.displayFrameRate

Description:

This attribute describes the display frame rate of the specialized processor.

Domain:

Integer

7. AdapterCommRes ()

(adapterCommResId,
averageDataUnitSize,
dataUnitInterarrivalDist,
numberOfDataUnitsDropped)

Identifiers:

* [adapterCommResId]

Description:

This object represents the communication resource that is of the adapter type used for digital transmission. Attributes of QoS interest include the average data unit size, the distribution of the incoming data units, and the number of data units dropped by the node.

7.1 AdapterCommRes.adapterCommResId

Description:

This attribute uniquely identifies the AdapterCommRes object.

Domain:

Id

7.2 AdapterCommRes.averageDataUnitSize

Description:

This attribute describes the average size of the data units of the adapter communication resource.

Domain:

Real

7.3 AdapterCommRes.dataUnitInterarrivalDist

Description:

This attribute describes the interarrival distribution of the data units.

Domain:

String

7.4 AdapterCommRes.numberOfDataUnitsDropped

Description:

This attribute describes the number of data units dropped.

Domain:

Integer

8. ModemCommRes ()

(modemCommResId,
attenuation/gain,
noiseFigure,
bandwidth)

Identifiers:

* [modemCommResId]

Description:

This object describes modem communication resources that are used for analog transmission. Attributes of QoS interest for this type of object include

1. The noise figure, as the signal-to-noise (S/N) ratio of an ideal and noiseless device, divided by the S/N ratio at the output of an amplifier or a receiver
2. The bandwidth supported by the modem
3. The range or transmission distance that the modem covers
4. The signal gain or attenuation provided by the modem.

8.1 ModemCommRes.modemCommResId

Description:

This attribute uniquely identifies ModemCommRes object.

Domain:

Id

8.2 ModemCommRes.attenuation/gain

Description:

This attribute specifies signal attenuation or gain in dB.

Domain:

Real

8.3 ModemCommRes.noiseFigure

Description:

This attribute specifies the noise figure for a transmission device in dB.

Domain:

Real

8.4 ModemCommRes.bandwidth

Description:

This attribute specifies the bandwidth supported by a transmission device.

Domain:

Integer

9. SwitchCommRes ()

(switchCommResId,
jitter,
dataUnitErrorRatio)

Identifiers:

* [switchCommResId]

Description:

This object represents hardware communication resources that belong to devices functioning as a switch. Since a switch processes fixed length data units, an attribute related to timeliness that is normally associated with a switch is jitter, which specifies the maximum variation in processing delay of a fixed length data unit.

9.1 SwitchCommRes.switchCommResId

Description:

This attribute uniquely identifies the SwitchCommRes object.

Domain:

Id

9.2 SwitchCommRes.jitter

Description:

This attribute provides the data unit delay variation incurred by the SwitchCommRes.

Domain:

Real

9.3 SwitchCommRes.dataUnitErrorRatio

Description:

This attribute specifies the ratio of data units in error over the data units sent across the SwitchCommRes.

Domain:

Real

10. RouterCommRes ()

(routerCommResId,
numberOfHops,
priorityProtocol,
routingAlgorithm)

Identifiers:

* [routerCommResId]

Description:

This object represents hardware communication resources of devices functioning as a router. Attributes related to QoS include the number of hops a data unit has to travel from the current node to reach the destination node, the priority applied to a routed protocol, the routing algorithm which provides some indication of CPU, memory, or bandwidth usage requirements.

10.1 RouterCommRes.routerCommResId

Description:

This attribute uniquely identifies the RouterCommRes.

Domain:

Id

10.2 RouterCommRes.numberOfHops

Description:

This attribute specifies the cumulative number of routers a particular data unit traverses, including the current router node.

Domain:

Integer

10.3 RouterCommRes.priorityProtocol

Description:

This attribute describes the priority applied to a routed protocol.

Domain:

String

10.4 RouterCommRes.routingAlgorithm

Description:

This attribute describes the routing algorithm used.

Domain:

String

11. CommProtocol ()

(commProtocolId,
communicationResourceId,
type)

Identifiers:

* [commProtocolId]

Description:

This object represents a set of conventions referred to as a protocol, which may be defined as a set of rules governing the exchange of data between two entities (anything capable of sending or receiving information).

The protocol type includes

1. Communication network protocols that address a number of or all of the layers specified in the OSI reference model. Examples include
 - Seven layers: SNA, APPLETALK
 - Layers 1-3: SNA, APPLETALK, TCP/IP, NOVELL IPX
 - Layers 1-2: ATM, FDDI, IEEE 802, X.25
 - Layer 1: DS-1/T1, DS-3/T3, RS-232, I.340, I.341.
2. Routing protocols such as RIP and OSPF.
3. Routing algorithms such as link state and distance vector algorithms.
4. Network management protocols such as SNMP and CMIP.

11.1 CommProtocol.commProtocolId

Description:

This attribute uniquely identifies the CommProtocol object.

Domain:

String

11.2 CommProtocol.communicationResourceId

Description:

This attribute formalizes the relationship "CommunicationResource supports CommProtocol."

Domain:

Id

11.3 CommProtocol.type

Description:

This attribute describes the type of protocol: network protocol, routing protocol, routing and vector algorithm, or network management protocol.

Domain:

String

12. CommPort ()

(commPortId,
communicationResourceId,
interfaceType,
throughput,
capacity,
status)

Identifiers:

* [commPortId]

Description:

This object represents a port of a CommunicationResource via which communication between two nodes takes place. Thus, a port can consist of data signals, control signals and additional signals that provide information such as timing and status.

Attributes of QoS interest for this object include the type of physical interface supported (RS-232, RS-449, I.430, X.21, DS-1/T1, DS-3/T3, STS-1/OC-1, STS-3/OC-3 . . .); maximum data rate (or capacity); throughput; and port status.

12.1 CommPort.commPortId

Description:

This attribute specifies the Id of a port.

Domain:

Id

12.2 CommPort.communicationResourceId

Description:

This attribute formalizes the relationship "CommunicationResource consists of CommPort."

Domain:

Id

12.3 CommPort.interfaceType

Description:

This attribute describes the type of physical interface supported (e.g., RS-232, RS-449, X.21, DS-3).

Domain:

String

12.4 CommPort.throughput

Description:

This attribute specifies the average throughput experienced by a CommPort.

Domain:

Real

12.5 CommPort.capacity

Description:

This attribute specifies the maximum data rate supported by a CommPort.

Domain:

Real

12.6 CommPort.status

Description:

This attribute specifies the status of a CommPort (up/down/not connected).

Domain:

String

13. CommLink ()

(commLinkId,
capacity,
mediumType,
linkType,
linkConnectivity,
noisePower,
bitErrorRate,
propagationDelay,
throughput,
attenuation,
utilization)

Identifiers:

* [commLinkId]

Description:

This object describes the physical link that connects two adjacent nodes. A link can consist of one or more channels with each channel occupying a portion of the link capacity.

Attributes of interest to QoS management include:

1. Capacity
2. Type of transmission supported (full duplex, half duplex, or simplex)
3. Bit error rate experienced by the link
4. Propagation delay through the link as a function of the link medium
5. Signal attenuation experienced by the link.

13.1 CommLink.commLinkId

Description:

This attribute uniquely identifies a link between endpoint connections.

Domain:

Id

13.2 CommLink.capacity

Description:

This attribute specifies the capacity (in bits per second) supported by a link.

Domain:

Integer

13.3 CommLink.mediumType

Description:

This attribute specifies the link medium. Its domain includes

1. Air
2. Fiber optic
3. Wireline
4. Transponded.

Domain:

String

13.4 CommLink.linkType

Description:

This attribute specifies the type of link, e.g., simplex, full duplex, or half duplex.

Domain:
String

13.5 CommLink.linkConnectivity

Description:
This attribute specifies the link connectivity, including

1. Point-to-point
2. Multipoint
3. Broadcast

Domain:
String

13.6 CommLink.noisePower

Description:
This attribute specifies the thermal and radiated noise power of the link in dBW.

Domain:
Real

13.7 CommLink.bitErrorRate

Description:
This attribute specifies the bit error rate supported by the link.

Domain:
Real

13.8 CommLink.propagationDelay

Description:
This attribute specifies the link propagation delay in seconds.

Domain:
Real

13.9 CommLink.throughput

Description:
This attribute provides the link throughput in bits per second.

Domain:
Integer

13.10 CommLink.attenuation

Description:

This attribute specifies the link attenuation in dB.

Domain:

Real

13.11 CommLink.utilization

Description:

This attribute describes the percentage of total link capacity that is in use.

Domain:

Real

14. CommChannel ()

(commChannelId,
commLinkId,
type,
bandwidth)

Identifiers:

* [commChannelId]

Description:

This object describes the communication channel between two end devices' CommPorts. The channel is considered as a logical partitioning of an analog or digital link. The primary attribute associated with a channel is the bandwidth supported by the channel (in bits per second for digital channels and in Hz for analog channels).

14.1 CommChannel.commChannelId

Description:

This attribute uniquely identifies the communication channel between two communication devices.

Domain:

Id

14.2 CommChannel.commLinkId

Description:

This attribute uniquely identifies the link that contains the channel in use, formalizing the relationship "CommLink consists of CommChannel."

Domain:

Id

14.3 CommChannel.type

Description:

This attribute specifies the channel type (virtual or physical).

Domain:

String

14.4 CommChannel.bandwidth

Description:

This attribute specifies the channel bandwidth in bits per second.

Domain:

Real

Appendix B

RESOURCE MODEL RELATIONSHIP SPECIFICATION

RESOURCE MODEL RELATIONSHIP SPECIFICATION

The Resource Model Relationship Specification provides a textual description of each relationship shown in the Resource Model. In general, relationships are listed sequentially as R1, R2, R3 through Rn with accompanying descriptions.

“IsA” relationships are considered supertypes whereby subordinate type objects can be grouped together into a higher class of objects. For example, central processing units (CPUs) and Specialized Processors can be grouped into the higher or broader class of processing resource objects. For supertype relationships, such as R1, a listing of all of the subtype objects follows the relationship description.

Binary relationships involve only two objects and can be represented in three fundamental forms, depending on the number of instantiations of the object that are described by the relationship. For example, a one-to-one relationship implies that one instantiation of an object maps into only one instance of another object. Other forms include one-to-many and many-to-many. These binary relationships are depicted in the Resource Model with single or multiple arrowheads to indicate the singularity or multiplicity of the relationship.

Associative relationships are used in conjunction with an associative object to describe additional information about the relationship between two objects. Relationship R7 in the Resource Model provides a relevant example. R7 indicates that many communication ports may be interconnected, and the communication channel (associative object) provides additional information for each instance of a port to port connection.

The RMRS provides a complete textual description of all relationships in the Resource Model and is combined with the RMOS to fully document the Resource Model.

R1 (Supertype)

Description:

This relationship indicates that CPU and SpecializedProcessor together constitute ProcessingResource. CPU and SpecializedProcessor have common attributes shown in ProcessingResource, yet also have unique attributes, and thus need to remain separate as subtypes of ProcessingResource.

ProcessingResource Subtypes:

CPU

SpecializedProcessor

R2 (Supertype)

Description:

This relationship defines the hierarchy of the Resource object. Resources can be classified as processing, storage, or communication, each with unique attributes to define/specify QoS.

_Resource Subtypes:

ProcessingResource

StorageResource

CommunicationResource

R3 (Supertype)

Description:

This relationship indicates that the CommunicationResource object can be used for analog or digital transmission. The adapter communication resource (for digital transmission) and the modem communication resource (for analog transmission) are therefore subtypes of CommunicationResource.

CommunicationResource Subtypes:

AdapterCommRes

ModemCommRes

R4 (Binary)

Description:

This relationship identifies the communication protocols supported by the communication resource. This is of interest to GRM because the protocol supported affects the QoS of the communication resource (i.e., a protocol may have different error correction schemes, different latencies, etc.). This relationship is formalized by the attribute communicationResourceId, which refers to the communication resource that supports this protocol.

CommunicationResource supports one or more CommProtocol.

CommProtocol is supported by exactly one CommunicationResource.

R5 (Supertype)

Description:

This relationship indicates that AdapterCommRes can be either a switch or a router. SwitchCommRes and RouterCommRes are therefore subtypes.

AdapterCommRes Subtypes:

SwitchCommRes

RouterCommRes

R6 (Binary)

Description:

This relationship identifies the ports on a particular communication resource. This is of interest to GRM because the attributes of the port (i.e., physical interfaces supported, availability) affect the QoS achievable by the communication resource. This relationship is formalized by the attribute communicationResourceId, which refers to the communication resource to which this communication port belongs.

CommunicationResource consists of one or more CommPorts.

CommPort belongs to exactly one CommunicationResource.

R7 (Associative)

Description:

This relationship describes the association between two communicating communication ports. It is a many to many relationship because any instantiation of communication port can be connected to any number of other communication ports. The relationship is conditional because, if communication port is not in use, it will not be connected to any other communication ports. This many-to-many relationship is formalized by the CommChannel associative object, which describes the channel created by two or more connected communication ports.

For each unique (CommPort, CommPort) exists exactly one instance of CommChannel

Connects zero or more instances of CommPort.
Is connected to zero or more instances of CommPort.

R8 (Binary)

Description:

This relationship indicates that there may be many channels in a single communication link. This is of interest to GRM, since the attributes of the link affect the QoS of the channel. This relationship is formalized by the attribute commLinkId, which refers to the communications link to which this channel belongs.

CommLink consists of one or more CommChannels.

CommChannel is part of exactly one CommLink.

Appendix C

RATE-CONTROLLED STATIC-PRIORITY QUEUING DISCIPLINE

RATE-CONTROLLED STATIC-PRIORITY QUEUING DISCIPLINE

In this appendix, a discussion is provided of how jitter can be eliminated at the network layer by the use of buffers in a distributed fashion. The equations for calculating amounts of buffer space required by end nodes and intermediate nodes to absorb jitter are provided by Zhang and Ferrari.*

Jitter is the maximum variation in delays experienced by any two datagrams from the same connection traversing across an underlying communications network. Large jitter can cause disruption to the playout of continuous media (video, audio, image) at a receiving end-system node. Jitter can be eliminated by the use of buffers at the receiver in a nondistributed fashion. However, a distributed mechanism for controlling delay jitter can be used to distribute the buffer space requirements among intermediate nodes and receiving end-system node, thus reducing the total amount of buffer space required, compared to a nondistributed mechanism. In this subsection, we provide a discussion of the distributed mechanism that applies the Rate-Controlled Static-Priority Queuing (RCSPQ) service discipline at the network layer in a packet-switching network. This mechanism makes the distribution of buffer space requirements more uniform over an end-to-end connection.

The RCSPQ service discipline is implemented with a rate controller and a static-priority scheduler, as depicted in Figure C-1. The rate controller consists of a set of delay-jitter controlling regulators corresponding to each of the connections traversing the current node. Each regulator in a particular node is responsible for eliminating jitter generated on the path starting at the scheduler in the preceding node and ending at the regulator in the node of interest. The regulator performs the task of assigning each datagram an eligibility time upon its arrival and holds the datagram for

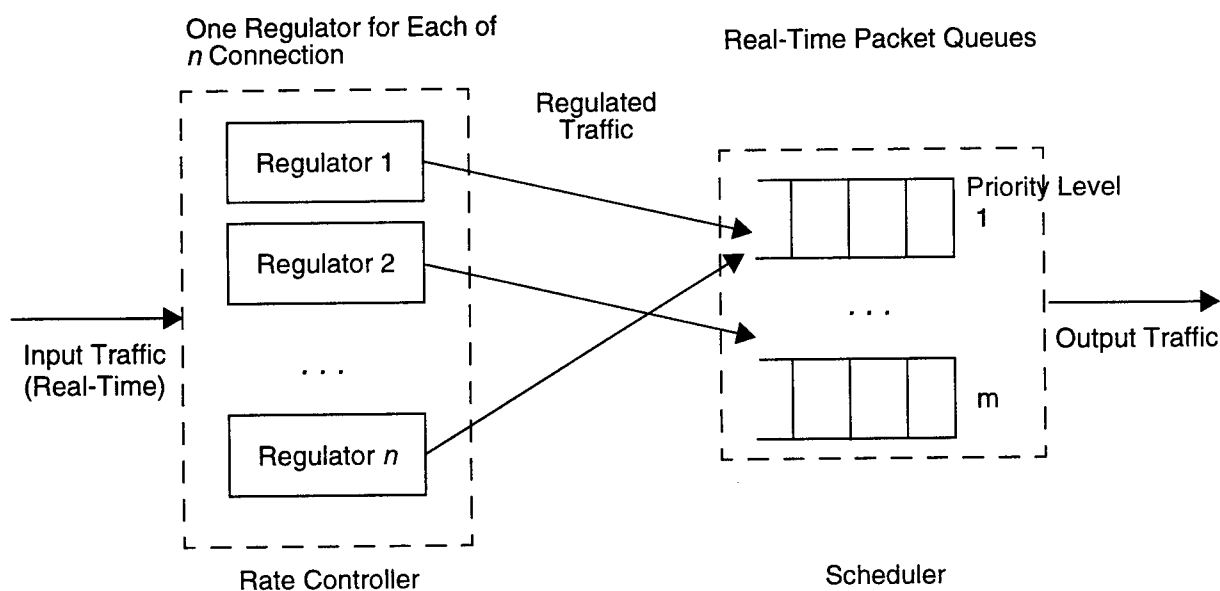


Figure C-1. Rate-Controlled Static-Priority Queuing

*Zhang, H., and D. Ferrari. 1992. "Rate-Controlled Static Priority Queuing," in *Proc. Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, California (November).

the assigned eligibility time before releasing it to the scheduler. In this way, all the datagrams from the same connection will experience the same delay by the time they enter the scheduler of the node where jitter absorption takes place.

The datagram eligibility time at the sending end node is the time a datagram enters the node's regulator. The eligibility time for subsequent nodes is the sum of the preceding node's datagram eligibility time, the delay bound for the datagram waiting time in the preceding node's scheduler, and the delay bound for the communication subnetwork between a node and its preceding node. The datagram waiting time includes both queuing time and datagram service time in the preceding node's Scheduler.*

Therefore, the original traffic pattern of the connection at the sending end node, represented by (X_{\min}, PDU_{\max}) , where X_{\min} is the minimum datagram inter-arrival time and PDU_{\max} is the maximum network layer datagram size on the connection, is fully reconstructed at the output of the regulator of each node along the end-to-end connection. This means that the jitter caused by the preceding node and its following communication subnetwork has been absorbed by the regulator in the current node. This distributed mechanism for controlling jitter also implies that the delay bound for the datagram waiting time in the scheduler of the receiving end node should be less than or equal to the end-to-end jitter bound specified at the network layer of the sending end node in order to provide end-to-end jitter guarantee.

The scheduler consists of a number of real-time datagram queues associated with different priority levels. A connection is assigned to a particular priority level at the connection's establishment time. All of the datagrams from the connection will be inserted into the real-time queue associated with that priority level. The scheduler chooses datagrams in first come first served (FCFS) order from the highest-priority nonempty queue.†

*The details on how to calculate delay bounds for the datagram waiting time in the scheduler of a node for each connection traversing the node can be found in Section 6 of the main text of this report.

†The method to determine buffer requirements at both intermediate nodes and receiving end-system nodes using a distributed jitter absorption mechanism is described in Section 6 of the main text of this report

DISTRIBUTION LIST

| addresses | number of copies |
|---|---------------------|
| AFRL/IFGA ATTN: TOM LAWRENCE 525 BROOKS ROAD ROME, NEW YORK 13441-4605 | 5 |
| SRI INTERNATIONAL 333 RAVENSWOOD AVE. MENLO PARK, CA 94025-3493 | 5 |
| AFRL/IFDIL TECHNICAL LIBRARY 25 ELECTRONIC PKY ROME NY 13441-4514 | 1 |
| ATTENTION: DTIC-DCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218 | 2 |
| ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714 | 1 |
| RELIABILITY ANALYSIS CENTER 201 MILL ST. ROME NY 13440-3200 | 1 |
| ATTN: GWEN NGUYEN GIDEP P.O. BOX 8000 CORONA CA 91718-8000 | 1 |

AFIT ACADEMIC LIBRARY/LDEE 1
2950 P STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765

WRIGHT LABORATORY/MTM, BLDG 653 1
2977 P STREET, STE 6
WRIGHT-PATTERSON AFB OH 45433-7739

ATTN: GILBERT G. KUPERMAN 1
AL/CFHI, BLDG. 248
2255 H STREET
WRIGHT-PATTERSON AFB OH 45433-7022

ATTN: TECHNICAL DOCUMENTS CENTER 1
DL AL HSC/HRG
2698 G STREET
WRIGHT-PATTERSON AFB OH 45433-7604

AIR UNIVERSITY LIBRARY (AUL/LSAD) 1
600 CHENNAULT CIRCLE
MAXWELL AFB AL 36112-6424

US ARMY SSDC 1
P.O. BOX 1500
ATTN: CSSD-IM-PA
HUNTSVILLE AL 35807-3801

TECHNICAL LIBRARY D0274(CPL-TS) 1
SPAWARSYSCEN
53560 HULL STREET
SAN DIEGO CA 92152-5001

NAVAL AIR WARFARE CENTER 1
WEAPONS DIVISION
CODE 49L0000
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

SPACE & NAVAL WARFARE SYSTEMS CMD 2
ATTN: PMW163-1 (R. SKIAND)RM 1044A
53560 HULL ST.
SAN DIEGO, CA 92152-5002

SPACE & NAVAL WARFARE SYSTEMS 1
COMMAND, EXECUTIVE DIRECTOR (PD134)
ATTN: MR. CARL ANDRIANI
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

COMMANDER, SPACE & NAVAL WARFARE 1
SYSTEMS COMMAND (CODE 32)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

CDR, US ARMY MISSILE COMMAND 2
REDSTONE SCIENTIFIC INFORMATION CTP
ATTN: AMSMI-RO-CS-R, DPCS
REDSTONE ARSENAL AL 35898-5241

ADVISORY GROUP ON ELECTRON DEVICES 1
SUITE 500
1745 JEFFERSON DAVIS HIGHWAY
ARLINGTON VA 22202

REPORT COLLECTION, CIC-14 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

AEDC LIBRARY 1
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUITE C211
ARNOLD AFB TN 37389-3211

COMMANDER 1
USAFSC
ASHC-IMD-L, BLDG 51801
FT HUACHUCA AZ 85613-5000

US DEPT OF TRANSPORTATION LIBRARY 1
FB104, M-457, RM 930
800 INDEPENDENCE AVE, SW
WASH DC 22591

AWS TECHNICAL LIBRARY 1
359 BUCHANAN STREET, RM. 427
SCOTT AFB IL 62225-5118

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

SOFTWARE ENGINEERING INSTITUTE 1
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVENUE
PITTSBURGH PA 15213

NSA/CSS 1
K1
FT MEADE MD 20755-6000

ATTN: DM CHAUHAN 1
COMC WICHITA
271 WEST THIRD STREET NORTH
SUITE 6000
WICHITA KS 67202-1212

PHILLIPS LABORATORY 1
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

DUSD(P)/DTSA/DUTO 2
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202