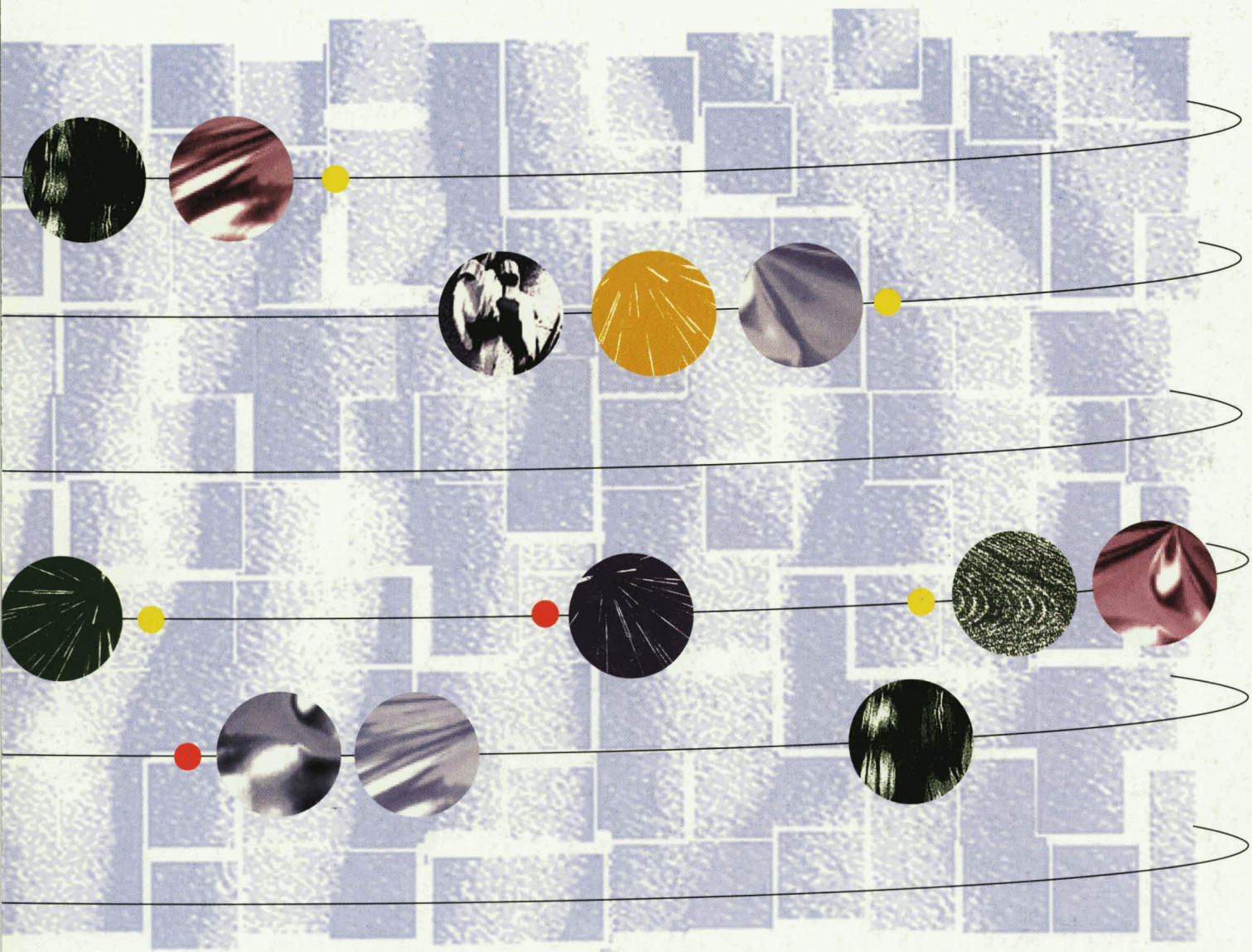
- *RAID Array Controllers*
- *Workflow Models*
- *PC LAN and System Management Tools*

Digital Technical Journal

Digital Equipment Corporation



Volume 6 Number 4
Fall 1994

The *Digital Technical Journal* is a refereed journal published quarterly by Digital Equipment Corporation, 30 Porter Road LJO2/D10, Littleton, Massachusetts 01460. Subscriptions to the *Journal* are $40.00 (non-U.S. $60) for four issues and $75.00 (non-U.S. $115) for eight issues and must be prepaid in U.S. funds. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Orders, inquiries, and address changes should be sent to the *Digital Technical Journal* at the published-by address. Inquiries can also be sent electronically to dtj@digital.com. Single copies and back issues are available for $16.00 each by calling DECdirect at 1-800-DIGITAL (1-800-344-4825). Recent back issues of the *Journal* are also available on the Internet at http://www.digital.com/info/DTJ/home.html. Complete Digital Internet listings can be obtained by sending an electronic mail message to info@digital.com.

Digital employees may order subscriptions through Readers Choice by entering VTX PROFILE at the system prompt.

Comments on the content of any paper are welcomed and may be sent to the managing editor at the published-by or network address.

ISSN 0898-901X

Documentation Number EY-T118E-TJ

Book production was done by Quantic Communications, Inc.

Cover Design

Our cover design is inspired by a system management topic in this issue. ManageWORKS software is a system and network management tool that presents an object-oriented, graphical view of a heterogeneous LAN environment. The multicolor circles on the cover represent the diverse objects, or entities, on the networks among which a system administrator "navigates" using the integrated components of the tool.

The cover was designed by Lucinda O'Neill and Joe Pozerycki, Jr., of Digital's Design Group.

Contents

Editor's Introduction

Jane C. Blake
Managing Editor

Three computing topics are presented in this issue of the *Journal:* a storage array controller for open system environments, workflow architectures and tools, and PC and LAN system management products.

The opening paper, by Steve Sicola, describes Digital's new HS series of StorageWorks array controllers. Designed for open systems, the controllers interface to host computers by means of the industry-standard SCSI-2 interconnect, as well as Digital's CI and DSSI host interconnects. Equally important to designers as openness were controller availability and performance. Innovative features were introduced, including dual-redundant controllers and Parity RAID firmware to ensure high availability, and a write-back cache that significantly improves performance. The paper concludes with a description of the common controller processing core for the SCSI, CI, and DSSI controller variants.

Workflow is the subject of two papers with differing perspectives. Christoph Bußler opens his paper with introductory definitions and implications of workflow concepts. He argues that a workflow that uses roles for task assignment is limited, especially in large, international enterprises. He states that by adding the dimension of organizational dependencies for task assignment a complex workflow is more precisely expressed. Using the example of a travel expense reimbursement workflow, Christoph shows how the Policy Resolution Architecture design principles support enterprise-level workflow deployment—reusability, security, generality, dynamics, and distribution. He also discusses the Policy Definition Language that formally describes workflow elements.

A second paper about workflow presents a tool, called DECmodel for Windows, for the development of business process models and their graphical presentation. Stew Hoover and Gary Kratkiewicz explain the reasoning behind the creation of a presentation layer in DECmodel that provides a graphical view of the business process while hiding the technical details of the model. The authors also cover implementation details, including the decisions to move from the original LISP environment to a C++ programming environment and to implement the knowledge base for DECmodel in ROCK, a frame-based knowledge representation.

We then shift the focus to ManageWORKS and POLYCENTER tools that have been developed to simplify the increasingly complicated job of system management. The first of three papers describes the development of the ManageWORKS Workgroup Administrator software. Dennis Giokas and John Rokicki discuss the design principles adopted for this product that enables system and network management of heterogeneous LANs from a single PC running Microsoft Windows. Key design elements are plug-in, customizable modules for system navigation and management, and the user interface framework, which controls the flow between modules. The authors offer scenarios to illustrate interactions between components.

Managing OpenVMS systems from a PC running the Microsoft Windows operating system can be accomplished with the OpenVMS Management Station, of which ManageWORKS is a key component. Jim Johnson defines the need for this scalable and secure client-server tool in OpenVMS environments, which can be clustered, distributed, expanded, and networked extensively. After a discussion of design alternatives, Jim describes the functions of the Station's client, communication, and server components.

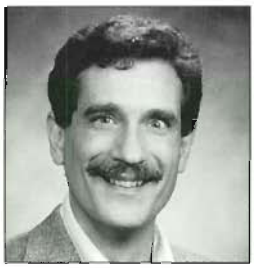
The final paper is about an initial system load (ISL) capability for automatic, network-directed, operating system software upgrades. John Lawson reviews goals for the POLYCENTER Software Distribution layered product, compares the POLYCENTER ISL process with the OpenVMS ISL process, and steps through the requirements for expanding the POLYCENTER Software Distribution capability to other platforms and operating systems.

Our next issue will celebrate the *Journal*'s tenth anniversary of publishing the technical achievements of Digital's engineers and partners. The issue will feature database technologies and new Alpha workstations and high-end server systems.

Jane Blake

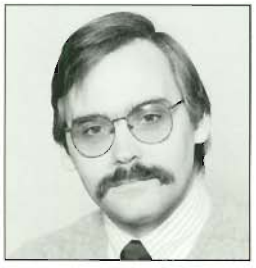
Biographies

Christoph J. Bußler Christoph Bußler is a faculty member at the Technical University of Darmstadt, Germany, where he is pursuing a Ph.D. degree. His research is in workflow and organization modeling, with a focus on organizational embedding of workflow management, and in architectures for enterprise-wide deployment of workflow management systems. While at Digital from 1991 to 1994, Christoph developed the Policy Resolution Architecture and its prototype implementation. He holds an M.C.S. (1990) from the Technical University of Munich and has published many papers on workflow management and enterprise modeling.



Dennis G. Giokas Dennis Giokas is currently a senior associate with Symmetrix, Inc. While at Digital from 1984 to 1995, he was a consulting engineer in the PATHWORKS group. He co-led PATHWORKS V5.0 and architected the user interface and system management tools. He was also architect and manager for the PC DECwindows program. Previously, Dennis worked at Arco Oil & Gas and The Foxboro Company developing process control software. He holds a Bachelor of Music from the University of Massachusetts at Lowell, a Master of Music from the New England Conservatory, and an M.S.C.S. from Boston University.

Stewart V. Hoover Employed at Digital Equipment Corporation between 1984 and 1994, Stew Hoover is currently an independent consultant specializing in modeling and simulation. Before joining Digital, he was an associate professor of industrial engineering and information systems at Northeastern University. Stew contributed to the development of the DECalc-PLUS application, Statistical Process Control Software (SPCS), and the DECwindows version of Symmod. He has written many papers and articles on simulation and is coauthor of *Simulation, A Problem-Solving Approach,* published by Addison-Wesley.



James E. Johnson A consulting software engineer, Jim Johnson has worked in the OpenVMS Engineering Group since joining Digital in 1984. He is currently a member of the OpenVMS Engineering team in Scotland, where he is a technical consultant for transaction processing and file services. His work has spanned several areas across OpenVMS, including RMS, the DECdtm transaction services, the port of OpenVMS to the Alpha architecture, and OpenVMS system management. Jim holds one patent on commit protocol optimizations. He is a member of the ACM.

Gary L. Kratkiewicz Gary Kratkiewicz is currently a scientist in the Intelligent Systems R&D Group at Bolt Beranek and Newman Inc. As a principal engineer in Digital's DECmodel engineering group from 1991 to 1994, Gary coordinated the architecture and high-level design specifications, and developed the knowledge base, script engine, API, and several user interface modules. Earlier at Digital, he developed an expert system for shipping and was project leader for a knowledge-based logistics system. Gary holds an S.B.M.E. from MIT and an M.S. in manufacturing systems engineering from Stanford University.

John R. Lawson, Jr. John Lawson joined Digital in 1984. He has been a member of the OpenVMS VAX Development Group and the POLYCENTER Software Distribution Development Group. His code exists in several layered products and in the OpenVMS VAX and OpenVMS AXP operating systems. He holds a B.M. degree from the Eastman School of Music (1984) and a B.S. in software engineering from the University of Rochester (1986). He is currently pursuing an M.S. in mathematics and computer science from the Colorado School of Mines. John has a U.S. patent pending for a unique sorting algorithm.

John C. Rokicki John Rokicki, the project leader for ManageWORKS Workgroup Administrator, is a principal software engineer within Digital's Network Operating Systems engineering organization. His primary responsibility is the design and implementation of the base services of the ManageWORKS product. Before joining Digital in 1990, he was employed by Data General Corp. and Sytron Inc. John holds a B.S. (1989) in computer science from Worcester Polytechnic Institute.

Stephen J. Sicola Consulting engineer Stephen Sicola is a member of the the Array Controller Group in the Storage Business Unit. He is working on the next generation of controllers and was the technical leader for the current StorageWorks controller product set. In earlier work, Steve developed software and hardware for such products as the HSC, KDM70, and advanced development controller projects. Steve joined Digital in 1979 after receiving a B.S.E.E. from Stanford University. He received an M.S.C.E. from the National Technological University in 1992.

Stephen J. Sicola |

The Architecture and Design of HS-series StorageWorks Array Controllers

The HS series of StorageWorks array controllers is a new family of Digital products that includes models for both open systems and systems that use Digital's proprietary buses. The HS-series controllers combine performance, availability, and reliability in total storage subsystem solutions that use industry-standard storage devices. The architecture and design of StorageWorks array controllers represents a balance between the market requirements and the available technology. The engineering trade-offs led to an innovative design that incorporates product features such as a dual-active controller configuration, write-back caching, Parity RAID technology, and SCSI-2 device handling.

The HS series of StorageWorks array controllers, a new addition to Digital's storage subsystem family, supports an open systems environment by allowing the attachment of industry-standard Small Computer Systems Interface (SCSI-2) devices to the controller.[1] Moreover, these controller products yield high availability and high performance. This paper describes the architecture and the design of the HSJ30, HSJ40, HSD30, and HSZ40 StorageWorks array controllers. These controllers interface to host computers by means of existing Digital interconnects, i.e., the Computer Interconnect (CI) and the Digital Storage System Interconnect (DSSI), as well as a SCSI-2 host interconnect to VAX, Alpha, and most other computers in the industry. The paper documents the design and development trade-offs and describes the resulting controllers and their features.

StorageWorks array controllers represent a significant change from Digital's original Hierarchical Storage Controller (HSC) subsystem, the HSC50 controller, which was designed in the late 1970s, and also from other Digital controllers such as the HSC60, HSC70, HSC90, and KDM70 controllers. The StorageWorks controllers discussed in this paper were designed to meet the following product goals:

1. Open systems capability. The goals for open systems capability were to use industry-standard storage devices attached to the controllers and to use an industry-standard host interconnect for one controller model. Using industry-standard devices would provide investment protection for customers because they would not have to change devices when a new controller was introduced or when they changed controller modules to use a different host interconnect. Industry-standard devices would also reduce overall subsystem cost because of the competitive nature of the storage device industry. The long-term use of both Digital and non-Digital devices was desired to provide a wide variety of device choices for customers. The use of an industry-standard host interconnect would allow StorageWorks controllers to be used with Digital and non-Digital host computers, further expanding the open systems capability. The SCSI-2 interconnect was chosen as the device interface and the host interface over other industry-standard interconnects for cost and strategic reasons.

2. High availability. The goals for high availability included both controller fault tolerance and storage (disk configuration) fault tolerance.

   Controller fault tolerance was achieved by developing a dual-redundant controller configuration in combination with new StorageWorks enclosures that provide redundant power supplies and cooling fans. The goal of the dual-redundant configuration was to have the surviving controller automatically assume control of the failed controller's devices and provide I/O service to

them. As a side benefit, such a configuration would provide load balancing of controller resources across shared device ports.

The storage fault-tolerance goal was to develop firmware support for controller-based redundant array of inexpensive disks (RAID).[2] The initial Parity RAID implementation incorporated the best attributes of RAID levels 3 and 5. The design provided the basis for later implementations of other forms of RAID technology, notably mirroring. Parity RAID supports the goal of storage fault tolerance by providing for continued I/O service from an array of several disks in the event that one disk fails. StorageWorks packaging that provides redundant power supplies and cooling should be combined with the Parity RAID technology to extend storage fault tolerance.

3. High performance. The goals for high performance were to specify controller throughput (the number of I/O operations per unit of time), latency (responsiveness), and data transfer rate (controller bandwidth) for each of the three controller platforms: CI, DSSI, and SCSI. The throughput was specified in the maximum number of read and write requests executed per second. The controllers had to speed up the response time for host I/O operations and thus deliver data with lower command latency than the HSC controllers. StorageWorks controllers had to achieve the highest possible data transfer rate and were to do so on a common platform.

   The platform-specific controller throughput goals were as follows. The initial goal for the CI-to-SCSI controller was 1,100 read requests per second; the long-term goal was 1,500 to 1,700 read requests per second. The initial goal for the DSSI-to-SCSI controller was 800 read requests per second; the long-term goal was 1,300 read requests per second. The initial goal for the SCSI-to-SCSI controller was 1,400 read requests per second; the long-term goal was 2,000 read requests per second. The controller throughput for write operations was slightly lower.

   To reduce latency, the controller hardware and firmware implemented controller I/O request caching. Designers initially decided to include 16 to 32 megabytes (MB) of cache memory on a separate optional cache module. Read caching was the beginning goal for the project; however, write-back caching was added during product development as a result of RAID technology investigations.

   Another approach to reduce latency was to develop controller-based disk striping, i.e., implement the RAID level 0 technology.[2] Specific goals were to achieve parallel access to all RAID level 0 array members for read and write operations and to streamline firmware to increase RAID level 0 performance.

   The Parity RAID performance goal was to overcome the well-known weaknesses of RAID level 3 (i.e., poor transaction throughput) and RAID level 5 (poor small-write performance) and to approach RAID level 0 striped array performance for both small and large read and write requests.[2] A combination of hardware-assisted parity computations and write-back caching helped achieve this goal. Parity calculations in hardware reduced firmware overhead to complete RAID level 5 write operations. Write-back caching minimized the effects of the RAID level 5 small-write penalty.[3] To meet the needs of customers who require high data transfer rates with RAID, RAID level 3–style algorithms must be added for the Parity RAID design.

   A common controller processing core had to be architected and designed to meet the performance needs of all the planned StorageWorks controllers (based on host interface capabilities). The platform had to execute the same base firmware, coupling new host interface firmware to the specific platforms. A common platform was believed to ease product development and to maximize reuse of firmware for the same "look and feel" in all products.

Open Systems Capability

For StorageWorks controllers to enter the open systems market, product designers had to consider the following aspects of open systems in the controller definition: the use of industry-standard device interconnects and industry-standard devices attached to the controller, and the use of industry-standard and Digital host interconnects.

SCSI-2 Device Interconnect

The SCSI-2 interconnect was chosen for the device interconnect because of its wide acceptance in the computer industry. During the controller definition phase, the StorageWorks packaging group was

concurrently designing and building storage device enclosures called shelves that would house up to seven 3.5-inch devices or two 5.25-inch devices. These shelves, connected to the controller, would allow a wide variety of SCSI-2 devices to be incorporated and would do so at a low cost because of the widespread use of SCSI-2 as a device interconnect.

StorageWorks controllers were designed to support the following types of SCSI-2 devices:

- Disk—rotating spindle disk drives and solid-state disks
- Tape—individual tape drives, tape loaders, and jukeboxes that contain robotic access to multiple drives from a media library
- CD-ROM
- Optical—individual disks and jukeboxes that contain robotic access to multiple drives from a media library

StorageWorks Controllers in System Environments

The desire to produce a controller with an open system host interconnect was coupled with a commitment to protect the investments of existing Digital customers who currently use CI and DSSI host interconnects. The strategy was to produce CI, DSSI, and SCSI variants of the StorageWorks array controller, all based on a common platform. As in the selection of the device interconnect, the SCSI-2 host interconnect variant was chosen because of its widespread use and low cost.

The controllers for the CI, DSSI, and SCSI interconnects were named the HSJ30/HSJ40, the HSD30, and the HSZ40, respectively. The designations of "30" and "40" represent a code for the number of device ports attached to the controller. The HSJ30 and HSD30 controllers have three device ports each, whereas the HSJ40 and HSZ40 have six device ports each. The number of device ports selected for each controller type was based on (1) the overall capability of the host port interconnect to support the aggregate capability of a number of device ports and (2) the desire to amortize controller cost against as many attached devices as possible.

StorageWorks controller configurations depend on the controller host interface. Marked differences exist in the configurations supported by CI-based OpenVMS VAXcluster configurations, DSSI-based OpenVMS VAXcluster configurations, and SCSI-based configurations in OpenVMS, DEC OSF/1, and other industry system environments. The basic differences are the number of hosts connected and whether or not other storage devices can be on the same host interconnect as the controller and the other hosts.

The CI configuration supports up to 32 nodes per bus. Each node may be either a storage controller (i.e., an HSJ30, an HSJ40, or an HSC device) or a host computer (i.e., a VAX or an Alpha system).

The DSSI configuration supports up to 8 nodes per bus. Each node may be either a storage controller (i.e., an HSD30 or an HSD05), a storage element (e.g., an RF73 device), or a VAX or an Alpha host computer.

The SCSI configuration supports up to 8 targets per bus. The HSZ40 controller, with its standard SCSI-2 host interface, may be connected to Digital Alpha computers (i.e., DEC 3000 and DEC 7000/10000 computers running the DEC OSF/1 operating system), Sun Microsystems computers, Hewlett-Packard computers, and IBM computers. Digital qualifies the HSZ40 controller for operation with additional vendors' systems according to market demand.

High Availability

To meet the goals of controller and storage fault tolerance, the designers of StorageWorks controllers developed a number of scenarios from which the controller can be fault tolerant with respect to failures in controller or attached storage components. The first aspect of fault tolerance considered is that of controller fault tolerance; the second is configuration fault tolerance.

Controller Fault Tolerance

Designers achieved controller fault tolerance by investigating the common faults that the controller could tolerate without requiring extreme design measures and incurring high costs. The results of this investigation drove the design of what became the dual-redundant HS-series controller configuration. This configuration incorporates several patented hardware and firmware features (patent pending).

The following faults can exist within a StorageWorks array controller and the attached StorageWorks packaging and *do not* make host data unavailable:

- Controller failure. In a dual-redundant configuration, if one controller fails, all attached storage devices continue to be served. This is called

failover. Failover occurs because the controllers in a dual-redundant configuration share SCSI-2 device ports and therefore access to all attached storage devices. If failover is to be achieved, the surviving controller should not require access to the failed controller.

- Partial memory failure. If portions of the controller buffer and cache memories fail, the controller continues normal operation. Hardware error correction in controller memory, coupled with advanced diagnostic firmware, allows the controller to survive dynamic and static memory failures. In fact, the controller will continue to operate even if a cache module fails.
- Power supply or fan failure. StorageWorks packaging supports dual power supplies and dual fans. HS-series controllers can therefore be configured to survive a failure of either of these components.
- SCSI-2 device port failure. A failure in a single SCSI-2 device port does not cause a controller to fail. The controller continues to operate on the remaining device ports.

The controller must be able to sense the failures just listed in order to notify the host of a fault-tolerant failure and then to continue to operate normally until the fault is repaired. The designers deemed this feature vital to reducing the time during which a controller configuration must operate with a failure present.

Another requirement of fault-tolerant systems is the ability to "hot swap" or "hot plug" components, i.e., to replace components while the system is still operating and thus to not cause the system to shut down during repairs. The designers made the controller and its associated cache module hot swappable. That is, one controller in the dual configuration can be replaced without shutting down the second controller, and the second controller continues to service the requests of the attached hosts. This feature, coupled with the hot-swap capability of StorageWorks devices, creates highly available systems.

Dual-redundant Controller Configuration Like all StorageWorks components, HS-series controllers are packaged in StorageWorks shelves. The StorageWorks controller shelf contains a backplane that accommodates one or two controllers and their associated cache modules, as well as SCSI-2 device port connectors. The packaging is common to all system environments. HS-series controllers mounted in a single shelf may be combined in pairs to form a dual-redundant controller configuration (shown in Figure 1) in which both controllers can access the same set of devices.

Figure 2 shows two HS-series controllers installed in a StorageWorks controller shelf in

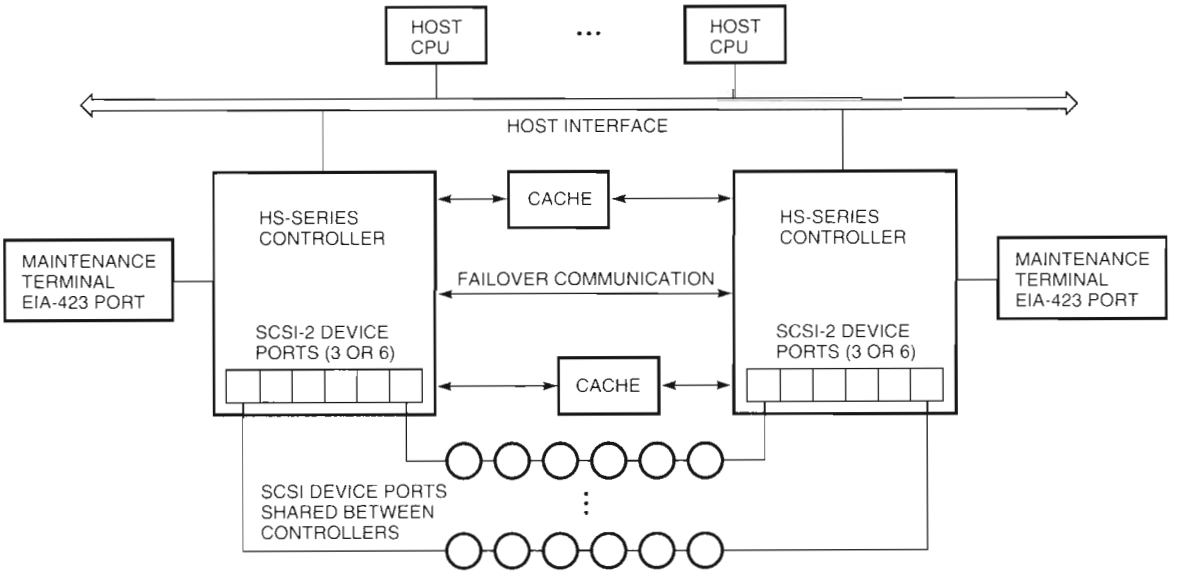


Figure 1 StorageWorks Controllers: System Block Diagram

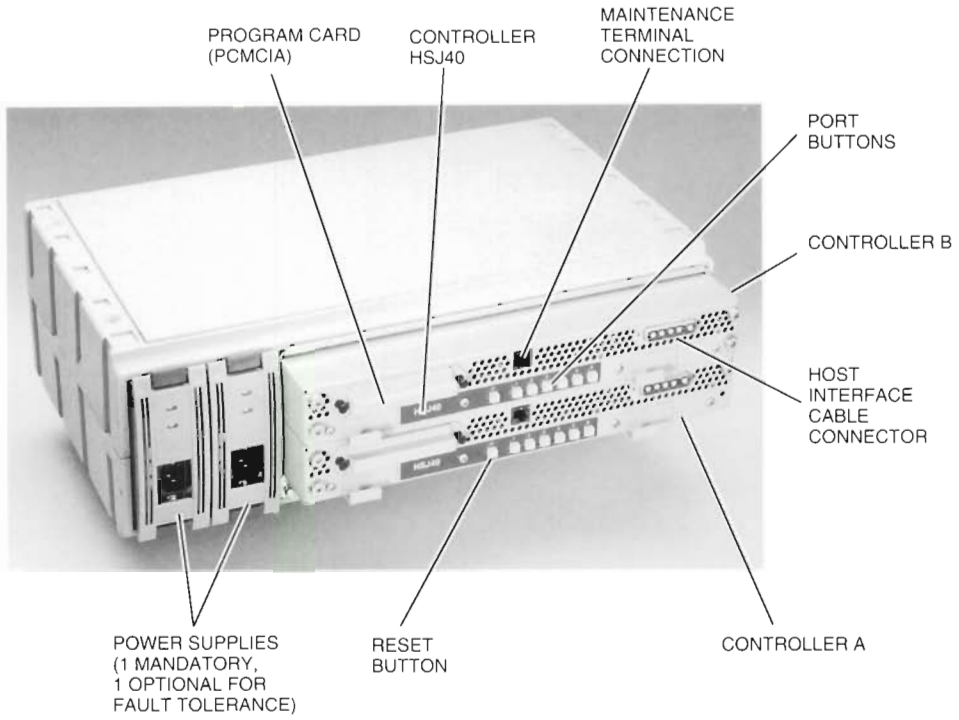


Figure 2 StorageWorks Controller Shelf

a dual-redundant configuration. Figure 3 shows two dual-redundant controller configurations mounted in a StorageWorks cabinet with several device shelves. The controllers connect to storage devices with cables that emerge from the controller shelf and attach to the device shelves.

The designers had to decide how the dual-redundant controller configuration could achieve high availability through fault tolerance. To meet the high-availability goals, the team addressed the concept of controller failover early in the design process. One fault-tolerant option considered was to run with a "hot-standby" controller that would become operational only if the main controller were to fail. A second option was to design a dual-active controller configuration in which two controllers would operate simultaneously. They would share and concurrently use device port buses (not devices), thus balancing the I/O load from host computers.

Both options allow for direct failover of devices without manual intervention. The hot-standby controller option requires either automatic configuration of the attached devices when the hot-standby controller becomes operational or nonvolatile (i.e., impervious to power loss) shared memory to hold the configuration information. The dual-active controller option requires that each controller have detailed knowledge about the other controller and the device state; it does not require that the controllers share a memory. The designers chose the second option because it provided load balancing and therefore potentially greater performance. However, they faced the challenge of designing a backplane and an interface between the controllers that would achieve the dual-active configuration but would not require a shared memory. The result of the design effort was the StorageWorks controller shelf.

StorageWorks Controller Shelf The StorageWorks controller shelf is an architected enclosure that allows a pair of StorageWorks controllers and their respective cache memory modules to be placed into the dual-redundant configuration, as shown in Figure 4. A cache module is attached to each controller for performance purposes. The controller shelf contains a backplane that includes intercontroller communication, control lines between the controllers, and shared SCSI-2 device ports. Since the two controllers share SCSI-2 device ports, the design enables continued device availability if one controller fails.

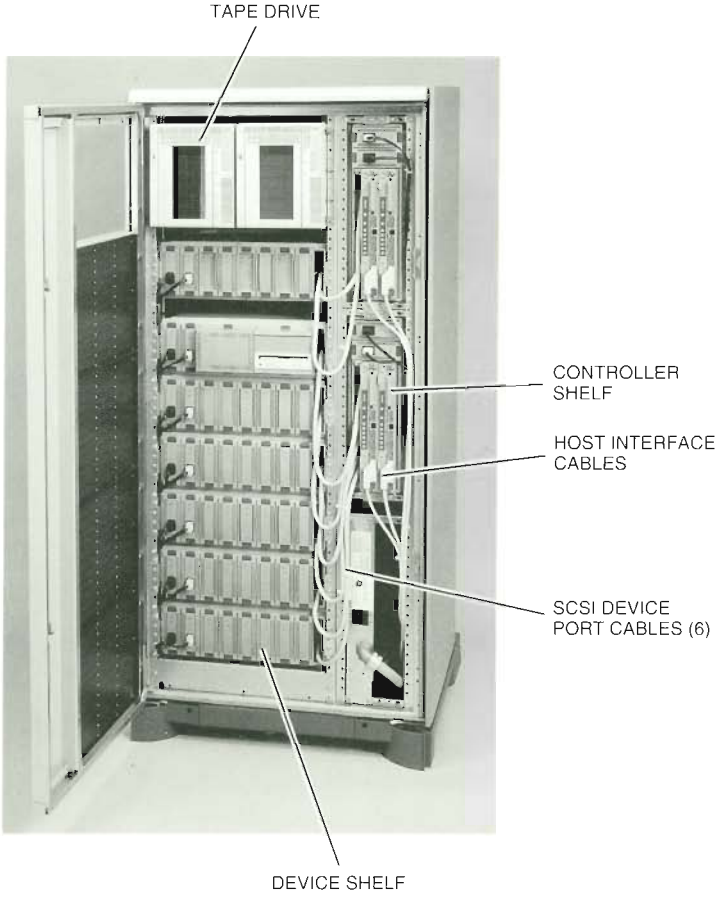
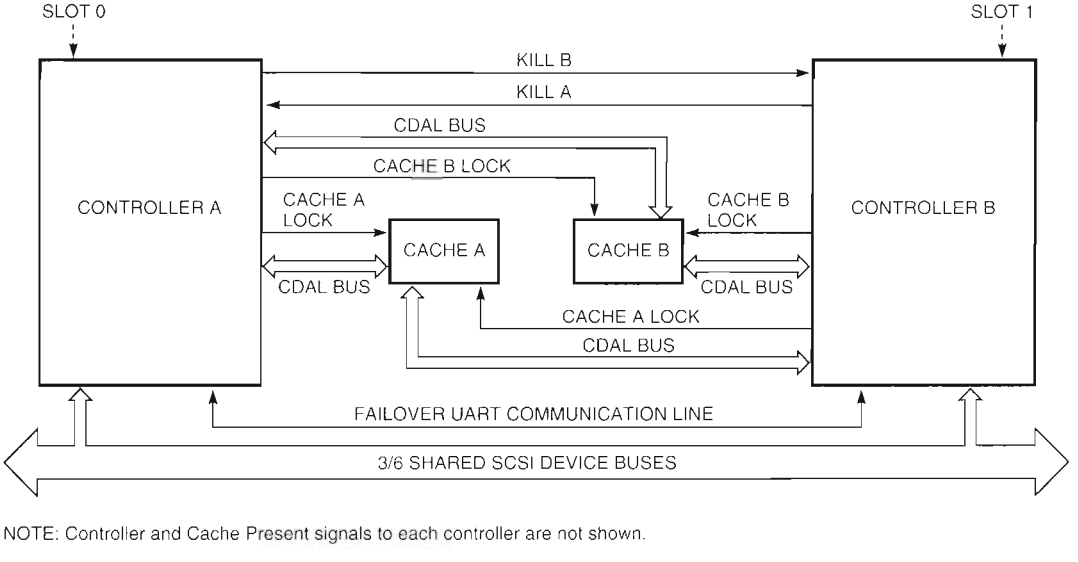


Figure 3 StorageWorks Cabinet



NOTE: Controller and Cache Present signals to each controller are not shown.

Figure 4 StorageWorks Controller Backplane: Controllers in a Dual-redundant Configuration

The backplane contains a direct communication path between the two controllers by means of a serial communication universal asynchronous receiver/transmitter (UART) on each controller. The controllers use this communication link to inform one another about

- Controller initialization status. In a dual-redundant configuration, a controller that is initializing or reinitializing sends information about the process to the other controller.
- "Keep alive" communication. Controllers send keep alive messages to each other at timed intervals. The cessation of communication by one controller causes a failover to occur once the surviving controller has disabled the other controller.
- Configuration information. StorageWorks controllers in a dual-redundant configuration have the same configuration information at all times. When configuration information is entered into one controller, that controller sends the new information to the other controller. Each controller stores this information in a controller-resident nonvolatile memory. If one controller fails, the surviving controller continues to serve the failed controller's devices to host computers, thus obviating shared memory access. The controller resolves any discrepancies by using the newest information.
- Synchronized operations between controllers. Specific firmware components within a controller can communicate with the other controller to synchronize special events between the hardware on both controllers. Some examples of these special events are SCSI bus resets, cache state changes, and diagnostic tests.

The other signals on the backplane pertain to the current state of the configuration within the controller shelf and to specific control lines that determine the operation of the dual-redundant controller configuration. The backplane state and control signals include

- Status about the presence of a controller's cache module. Each controller can sense the presence or absence of its cache to set up for cache diagnostics and cache operations.
- Status about the presence of a second controller, which indicates a dual-redundant configuration. Each controller can sense the presence or absence of the other controller in a dual-redundant configuration. This assists in controller setup of dual-controller operation as well as general controller initialization of the dual-redundant configuration.
- Status about the presence of the second controller's cache. Each controller can sense the presence or absence of the other controller's cache for dual-controller setup purposes.
- The "KILL" signal. In a dual-redundant configuration, each controller has the capability to use the KILL control signal to cause a hardware reset of the other controller. However, once one controller asserts the KILL signal, the other controller loses the capability. The KILL signal ensures that a failed or failing controller will not create the possibility of data corruption to or from attached storage devices.

  The KILL signal denotes that failover to the surviving controller should occur. A controller asserts the KILL signal when the other controller sends a message that it is failing or when normally scheduled keep alive communication from the other controller ceases. The KILL signal is also used when both controllers decide to reset one another, e.g., when the communication path has failed.

  The designers had to ensure that only one controller could succeed in the KILL operation, i.e., that no window existed where both controllers could use the KILL signal. After firmware on a controller asserts the KILL signal to its dual-redundant partner, the KILL recognition circuitry within the controller that asserted the signal is disabled. The probability of true simultaneous KILL signal assertion was estimated at 10^{-20}, based on hardware timing and the possibility of synchronous dual-controller operation.
- The cache LOCK signals. The cache LOCK signals control access to the cache modules. The dual-controller architecture had to prevent one controller from gaining access to a cache module that was being used by the other controller and had to allow the surviving controller to access the failed controller's cache. The access control had to be implemented in either firmware or hardware.

  A firmware solution would involve a software locking mechanism that the controllers would recognize and cooperatively use to limit cache module access to the associated controller. This

method had an inherent problem: firmware alone may not prevent inadvertent cache access by a failing controller. The designers therefore had to implement a hardware lock mechanism to prevent such inadvertent access.

The hardware lock mechanism was implemented with control signals from each controller. The signals are utilized by hardware to prevent inadvertent access and by firmware to limit cache module access to the associated controller. From each controller, the designers implemented two LOCK signals that extend individually to each cache module and are visible to both controllers. The cache LOCK signals are illustrated in Figure 4.

The LOCK signals allow a controller to achieve exclusive access to a specific cache module to ensure data integrity. LOCK signals from a controller that has been "killed" by its dual-redundant partner are reset so that the partner may fail over any unwritten cache data in the write-back cache.

Failover Controller failover is a feature of the dual-redundant configuration for StorageWorks controllers. Failover of a controller's devices and cache to the other controller occurs when

- A controller fails to send the keep alive message. This situation can occur because of a controller failure in the dual UART (DUART) or in any other non-fault-tolerant portion of the controller module. In this scenario, the surviving controller uses the KILL signal to disable the other controller, communicates to the failed controller's devices, and then serves the failed controller's devices to hosts.

  The failover of a controller's cache occurs only if write-back caching was in use before the controller failure was detected. In this case, the surviving controller uses the failed controller's cache to write any previously unwritten data to the failed controller's disks before serving these disks to hosts. When the surviving controller has written the data to disks (i.e., flushed the data), it releases the cache to await the failed controller's return to the dual-redundant configuration through reinitialization or replacement.

- A customer desires to change the load balance of one or more devices attached to one controller to the other controller. This specialized use of failover provides a load-balancing feature that the designers considered valuable in a dual-active controller configuration. Load balancing is static in the controller, i.e., devices are allocated to one controller or to the other, not shared dynamically. To change allocation, the system manager must change the preferred path of device access. This is accomplished by accessing either the maintenance port of the controller or the configuration firmware through the host interface (e.g., the diagnostics and utilities protocol for CI and DSSI systems).

- The cache module battery is low or has failed. This special case of failover is used in conjunction with Parity RAID operations for the reasons described in the Parity RAID technology portion of the following section. The main issue is to continue to provide as much data protection as possible for Parity RAID disk configurations when the battery on the write-back cache is low or bad.

- The controller is unable to communicate with the devices to which it is currently allocated for host operations. This situation can occur if a device port on a controller fails.

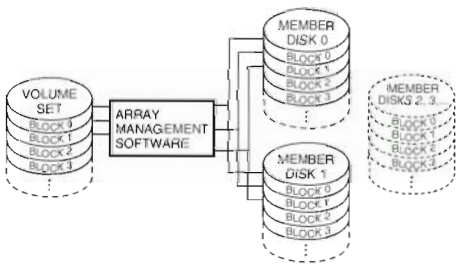
Storage Fault Tolerance

Storage fault tolerance is achieved by ensuring that power or environmental factors do not cause devices to be unavailable for host access and by using firmware to prevent a device failure from affecting host accessibility.

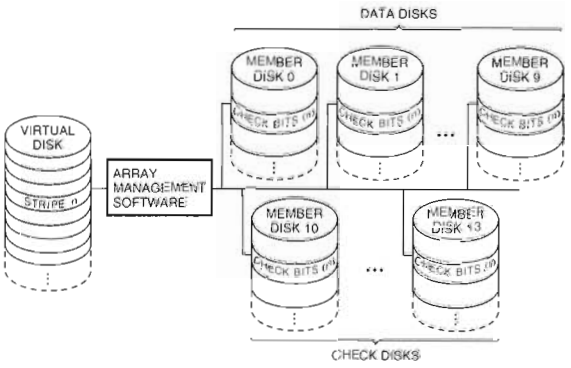
Environmental Factors StorageWorks enclosures provide for optional redundant power supplies and cooling fans to prevent power or fan failures from making devices unavailable. The SCSI-2 cables that connect device shelves to the controller shelf carry extra signals to alert the controller to power supply or fan failures so that these conditions may be reported to host computers. The enclosures must contain light-emitting diodes (LEDs) to allow a controller to identify failed devices. In addition, a cache module can fail, and the controller will continue to operate.

RAID Technology To prevent a device failure from affecting host access to data, the designers introduced a combined firmware and hardware implementation of RAID technology.[2] The designers had to decide which RAID level to choose and what type of hardware (if any) was required for the implementation.

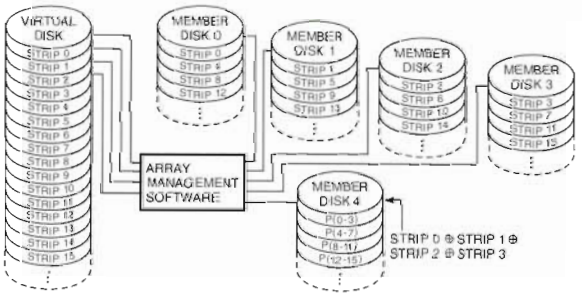
The designers considered RAID levels 1 through 5 as options for solving the problem of disk failures that affect data availability. RAID level 1 (disk mirroring, which is depicted in Figure 5a) was rejected because of its higher cost, i.e., the cost of parts to implement the mirroring.[2] Each disk to be protected implies an inherent cost of one additional housed, powered, and attached disk. RAID level 1 was also discounted because software-based solutions were available for many of the hosts for which the HS-series controllers were initially targeted.



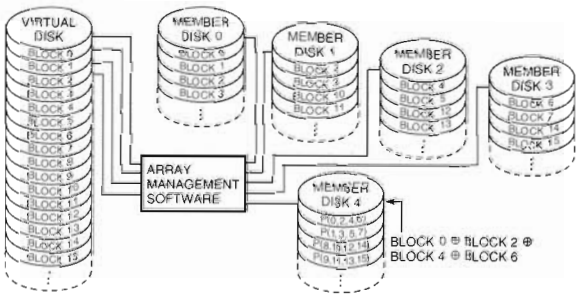
(a) Mapping for a RAID Level 1 Array



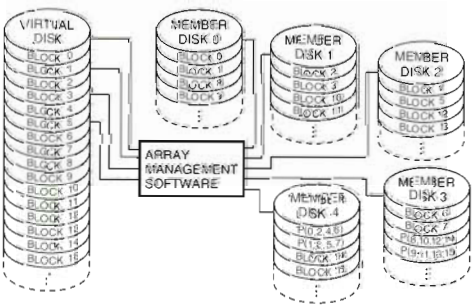
(b) Mapping for a RAID Level 2 Array



(c) Mapping for a RAID Level 3 Array



(d) Mapping for a RAID Level 4 Array



(e) A Typical Mapping for a RAID Level 5 Array

Figure 5 Mapping for RAID Levels 1 through 5

RAID levels 2 through 4, illustrated in Figures 5b through 5d, were rejected because they do not provide good performance over the entire range of I/O workloads for which the controllers were targeted.[4] In general, these RAID levels provide high, single-stream data transfer rates but relatively poor transaction processing performance.

RAID level 5 in its pure form was rejected because of its poor write performance, especially for small write operations.[2] The designers ultimately chose RAID level 5 data mapping (i.e., data striping with interleaved parity, as illustrated in Figure 5e) coupled with dynamic update algorithms and write-back caching to overcome the small-write penalty. This implementation is called Parity RAID.

An HS-series Parity RAID array appears to hosts as an economical, fault-tolerant virtual disk unit. A Parity RAID virtual disk unit with a storage capacity equivalent to that of n disks requires $n + 1$ physical disks to implement. Data and parity are distributed (striped) across all disk members in the array, primarily to equalize the overhead associated with processing concurrent small write requests.[2]

If a disk in a Parity RAID array fails, its data can be recovered by reading the corresponding blocks on the surviving disk members and performing a parity comparison (using exclusive-OR [XOR] operations on data from other members). Figure 6 illustrates this regeneration of data.[1]

HS-series controller developers overcame a number of challenges. Foremost among them was the elimination of the RAID level 5 write hole. Parity RAID with its RAID level 5 striping is susceptible to the RAID level 5 write hole. A write hole is data corruption that occurs when all the following events take place.

- A controller failure occurs with a host's write request outstanding.
- Either the updated data or the updated parity for the host's write request has been written to disk but not both.
- A failure of a different disk occurs after the controller failure has been repaired.

To eliminate this write hole, designers had to develop a method of preserving information about ongoing RAID write operations across power failures such that it could be conveyed between partner controllers in a dual-redundant configuration.

Designers decided to use nonvolatile caching of RAID write operations in progress.[5] Three alternatives were considered:

1. An uninterruptible power supply (UPS) for the controller, cache, and all attached disk devices. This choice was deemed to be a costly and unwieldy solution because of the range of possible requirements. The indeterminate amount of data in the cache to be written and the power consumption of a wide variety of devices would necessitate a very large backup power source to ensure enough time for all cached write data to be written to attached devices.

2. A battery in the controller and device enclosures (i.e., shelves) to allow enough time for the writing of cached data in the event of a power failure. StorageWorks device enclosures can accommodate either redundant power supplies or one power supply and one backup battery for configurations that do not require redundancy. There is no provision for both redundant power

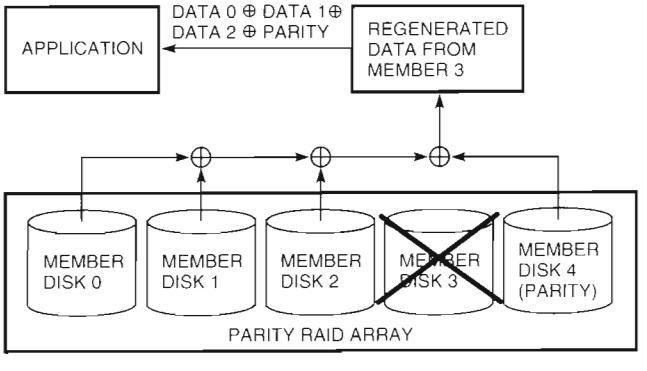


Figure 6 Regenerating Data in a Parity RAID Array with a Failed Member Disk

supplies and a battery. This conflict between fault-tolerant StorageWorks shelf configurations with dual power supplies and the desire to add a battery for write-back caching was unacceptable to the designers because of the loss of power redundancy to gain write-back cache integrity.

3. A controller-based nonvolatile cache. The options for controller-based nonvolatile caching included (a) a battery-protected cache for write data, (b) an additional nonvolatile random-access memory (NVRAM) on the controller to journal RAID writes, and (c) a battery-protected cache for both read and write data.

   With a battery-protected write cache, data must be copied if it is to be cached for subsequent read requests. Designers deemed the potential performance penalty unacceptable.

   Using controller NVRAM as a RAID write journal not only closes the RAID level 5 write hole but also provides a small write cache for data. This approach also requires data copying and creates an NVRAM access problem for the surviving controller to the failed controller NVRAM to resolve any outstanding RAID write requests.

   The third controller-based nonvolatile cache option, to battery-backup the entire cache, solved the copy issue of option 3a and the failover issue of option 3b.

The designers chose option 3c, the battery-protected read/write cache module. A totally nonvolatile cache had the advantage of not requiring write-cache flushing, i.e., copying data between the write cache and the read cache after the write data has been written to devices. Segregated cache approaches (part nonvolatile, part volatile) would have required either copying or discarding data after write-cache flushing. Such approaches would have resulted in a loss of part of the value of using the caching algorithm by allowing only read caching of read data already read. Another benefit of a nonvolatile read/write cache is failover of the cache module in the event of a controller failure. This further reduces the risk associated with a RAID level 5 write hole because unwritten write operations to Parity RAID arrays can be completed by the surviving controller after failover.

To achieve a total nonvolatile cache, the designers opted for the battery solution, using two 3-by-5-by-0.125-inch lead-acid batteries that supply up to 100 hours of battery backup for a 32-MB cache module. The batteries eliminated the need for a special (and costly) nonvolatile memory write cache and allowed data hold-up after power failure. The designers chose lead-acid batteries over NiCAD batteries because of their steady power retention and output over time. This option protects against most major power outages (of five minutes to five days) and all minor power outages (of less than five minutes). Most power outages (according to studies within Digital) last less than five minutes and are handled in the same manner as major outages, that is, by flushing write data immediately after power has been restored to the controller configuration. Battery status is provided to firmware, which uses this information to make policy decisions about RAID arrays and other virtual disk units with write-back caching enabled.

For an HS-series controller to support Parity RAID, its cache module must have batteries installed. The batteries make the cache nonvolatile and enable the algorithms that close the RAID level 5 write hole and permit the use of the write-back cache as a performance assist, both vital for proper Parity RAID operation. If the controller firmware detects a low- or bad-battery condition, write-back caching is disabled. The controller that detects the condition tries to fail over Parity RAID units to the other controller in the dual-redundant configuration to keep the units available to hosts. If the other controller cache module has a low- or bad-battery condition, the Parity RAID unit is made unavailable to hosts to protect against data loss or data corruption should a power failure occur. When the batteries are no longer low, Parity RAID units are again made available to hosts. Any Parity RAID units that had been failed over to the other controller would fail back, i.e., return, to the controller that originally controlled them. The module hardware and firmware support read caching regardless of the presence of a battery.

After solving the RAID level 5 write-hole problem, the designers decided to automate the Parity RAID recovery process wherever possible. This goal was adopted so that customers would not have to understand the technology details in order to use the technology in the event of a failure. StorageWorks controller firmware developers, therefore, chose to add automatic Parity RAID management features rather than require manual intervention after failures. Controller-based automatic array management is superior to manual techniques because the

controller has the best visibility into array problems and can best manage any situation given proper guidelines for operation.

An important feature of Parity RAID is the ability to automatically bring a predesignated disk into service to restore data protection as quickly as possible when a failure occurs. Other controllers in the industry mandate configurations with a hot-standby disk, i.e., a spare disk, dedicated to each Parity RAID unit. A hot-standby disk is powered and ready for firmware use if an active member disk of its Parity RAID unit fails. This concept is potentially wasteful since the probability that multiple Parity RAID units will have simultaneous single-member disk failures is low. The designers therefore had the options of making spare disks available on a per-Parity RAID unit basis or having a pool of spares, i.e., a spare set, that any configured Parity RAID unit could access. The designers chose the pool of spares option because it was simpler to implement and less costly for the customer, and it offered the opportunity to add selection criteria for spare set usage and thus maximize either performance or capacity efficiency.

To allow more flexibility in choosing spare set members, designers made two spare selection options available: best fit and best performance. The best fit option allows for disk devices of different sizes to compose the pool of spares. When a spare disk is needed after a member of a Parity RAID unit fails, the device with the best fit, that is, whose size most closely matches that of the failed disk (typically of the same size but possibly of greater capacity), is chosen. The best performance option can reduce the need for physical reconfiguration after a spare is utilized if a spare attached to the same device port as the failed array member can be allocated. The best performance option maintains operational parallelism by spreading array members across the controller device ports after a failure and subsequent spare utilization.

These features allow automatic sparing of failed devices in Parity RAID units and automatic reconstruction after a spare device has been added to the Parity RAID unit.[6] Furthermore, any drive of at least the size of the smallest member of a Parity RAID unit is a candidate spare, which reduces the need for like devices to be used as spares. (Typically, however, spare set members are like members.)

A Parity RAID unit with a failed member will become unavailable and lose data if a second failure occurs. The HS-series automatic sparing feature reduces the window of possible data loss to the time it takes to reconstruct one Parity RAID unit. Mean time between data loss (MTBDL) is a combination of the mean time to repair (MTTR) and the failure rate of a second device in a Parity RAID unit. The automatic sparing feature reduces the MTTR and thus increases the MTBDL. Data loss can occur only in the highly unlikely event that a failure occurs in another RAID set member before the reconstruction completes on the chosen spare. During Parity RAID reconstruction, the controller immediately makes the host read or write request to the reconstructing member redundant by updating parity and data on the spare after the host read or write operation. Parity RAID firmware quickly reconstructs the rest of the Parity RAID unit as a background task in the controller. If the member that is being reconstructed happens to fail and other spare set members remain, reconstruction on a new spare begins immediately, further reducing the probability of data loss.

Parity RAID member disk failure declaration is key to the efficient use of spares and to preventing unwarranted use of spares. If a write command to a RAID set member fails, RAID firmware assumes that the SCSI-2 disk drive has exhausted all internal methods to recover from the error. SCSI-2 disk drives automatically perform bad block replacement on write operations as long as there is space available within the disk drive revector area (the area where spare data blocks can be mapped to a failed block). The designers chose this method over more complex retry algorithms that might encounter intermittent failure scenarios. Empirical information related to previous storage devices showed that localized write failures are rare and that this strategy was sound for the majority of disk access failures.

When read failures occur, data is regenerated from the remaining array members, and a forced bad block replacement is performed. Metadata on the disk is used to perform this function atomically, that is, to perform the bad block replacement even if a power failure occurs during the replacement.[7] If the disk cannot replace the block, then the Parity RAID member disk is failed out and an attempt is made to choose a suitable spare from the spare set. If no spare is available, the Parity RAID unit operates in reduced mode, regenerating data from the failed member when requested by the hosts.[1]

Parity RAID firmware uses the metadata to detect a loss of data due to catastrophic cache failure, inappropriate device removal, or cache replacement without prior flush of write data. The designers

considered it important that the controller firmware be able to detect these data loss conditions and report them to the host computers.

The failure scenarios just described occur infrequently, and the StorageWorks Parity RAID firmware is able to recover after such failures. During a typical normal operation, the main challenge for Parity RAID firmware is to achieve a high level of performance during write operations and a high level of controller performance in general.

High Performance

As discussed earlier, the performance goals for the StorageWorks controllers were in the areas of throughput and latency. Bandwidth goals were based on the architecture and technology of the controller platform. The designers met the performance goals by producing a controller that had a low command overhead and that processed requests with a high degree of parallelism. The firmware design achieves low overhead by means of the algorithms running on the controller, coupled with RAID and caching technology. The hardware design that allows for low command overhead and high data transfer rates (bandwidth) is discussed in the section Common Hardware Platform.

Command Processing

The StorageWorks designers maximized the number of requests the controller can process per second by reducing the command processing latency within the controller firmware. The firmware utilizes controller-based caching and also streamlined command processing that allows multiple outstanding commands to be present in the controller.

To meet the varying needs of customer applications, the controller supports both Parity RAID and RAID level 0. The designers decided to include RAID level 0 as a controller feature because of its inherent parallelism, even though RAID level 0 is not fault tolerant without external redundancy.

StorageWorks controllers service all device types, but the designers felt that disk device performance was the key metric for determining how well a controller supports RAID technology. The controller firmware was designed to efficiently control individual devices (commonly referred to as "just a bunch of devices" [JBOD]) and Parity RAID, prioritizing requests to each of the SCSI-2 device ports on the controller. StorageWorks controllers comply with SCSI-2 protocols and perform advanced SCSI-2 functions, such as tagged queuing to all attached SCSI-2 storage devices for greater performance.[1]

Discussions of the RAID level 0 technology and of how the designers used Parity RAID technology to overcome some of the performance bottlenecks follow.

Striping—RAID Level 0

Digital has used RAID level 0 technology, that is, striping, in systems for at least five years, in its host computers using software as well as in its controllers. Striping allows a set of disks to be treated as one virtual unit. Device data blocks are interleaved in strips, i.e., contiguous sets of blocks, across all disks, which provides high-speed parallel data access. Figure 7 illustrates the mapping for a RAID level 0 array.[1] Since a striped disk unit inherently lacks fault tolerance (i.e., if one device in the set fails, data is lost), controller-based striping is typically used in conjunction with host-based mirroring or in cases where data can be easily reproduced. Stripe sets achieve high performance because of the potential for parallelism by means of the device and data organization. The key difference between RAID level 0 and RAID levels 3 and higher is that striping results in the interdependence of data written to different devices.

Controller Caching

Caching with StorageWorks controllers was originally read caching only. When the designers decided to use a nonvolatile cache to eliminate the RAID level 5 write hole, write-back caching on the controller became a viable option.

Controller Read Caching Read caching was intended to reduce latency in the controller by minimizing the need to access devices continuously for repeated host read requests to the same locations on attached devices. Read caching must also address the issue of how to handle write data for later use. The design could have incorporated on-board controller memory to hold write data. However, this would require copying the write data to the read cache after the write data had been written to the devices and would result in inefficient use of the read cache. Therefore, the designers decided to have the read cache serve as a write-through cache as well. Read caching would be disabled/enabled per logical unit presented to the host instead of having global read caching, where a logical unit is one or more devices configured as one virtual device.

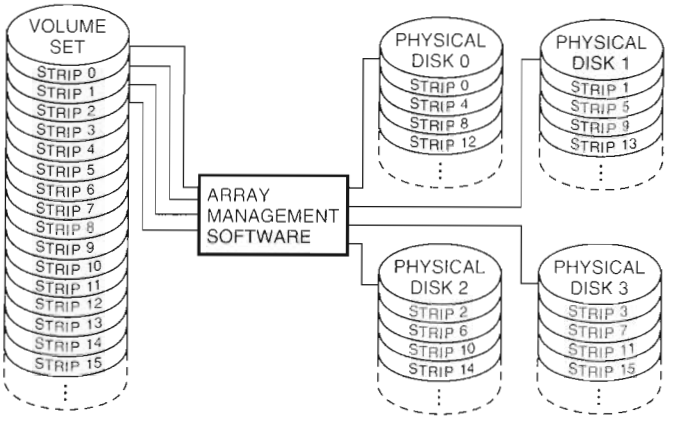


Figure 7 Mapping for a RAID Level 0 Array

Thus, customers can specify for which virtual devices they want caching enabled.

The read and write-through caching firmware receives requests from other parts of the controller firmware (e.g., a host port, a device port, and the Parity RAID firmware) and proceeds as follows.

For reads requests, the caching firmware provides

1. The data pointers to the cached request, i.e., the cache hit

2. The data pointers for part of the request, i.e., a partial cache hit, which means that the remaining data must be retrieved from the device or devices being requested

3. A status response of cache miss, which means that storage management must retrieve the data from the device or devices being requested

For write requests, the caching firmware offers the cache manager data from the request. The cache manager places the previous data pointers into the read cache tables after the data is written through the cache to the devices. Firmware tells the device port hardware where to find write data, and the port hardware transfers the data.

Read caching in the first version of the controller firmware allowed the controller to achieve the initial throughput goals across the three controller platforms. The current software version, version 2.0, was shipped in October 1994 and exhibits even greater throughput performance. Table 1 shows the I/O performance for the three StorageWorks controller platforms with read caching enabled.

Table 1 StorageWorks Controller I/O Performance with Read Caching

Controller	Read Requests per Second	Write Requests per Second
HSJ30/HSJ40	1,550	1,050
HSD30	1,000	800
HSZ40	2,250	1,500

Write-back Caching—Performance Aspects As noted earlier, when the cache module contains batteries, the memory is nonvolatile for up to 100 hours. The StorageWorks controller can use the nonvolatile cache to increase controller performance by reducing latency for write request Parity RAID performance to a level similar to that of RAID level 0 (simple disk striping). The controller can also utilize the write-back cache to reduce the latency of JBOD and RAID level 0 disk configurations. As with read caching, write-back caching is disabled/enabled per logical unit.

The write-back caching firmware controls the usage of both a surviving controller's cache module and a failed controller's cache module. When it receives a write request, the controller places the data in the cache, marks the request as complete, and writes the data based on internal controller firmware policies (write-back caching). To provide greater performance during Parity RAID operations than simple write-back caching could provide, the write-back cache firmware is also tied to the Parity RAID firmware.

In addition to dealing with the continual problem of controller latency on write commands, designers had to overcome the RAID level 5 small-write penalty with parity updates to RAID set members. Write-back caching was chosen over RAID level 3 hardware assists as a Parity RAID strategy because RAID level 3 does not provide a wide range of benefits for all customer workloads. Write-back caching provides latency reductions for RAID and non-RAID configurations. Write-back caching also increases write-request throughput. For example, the published performance numbers for write throughput with write-back caching enabled in version 2.0 firmware appear in Table 2.

The use of write-back caching resulted in a 20 to 30 percent increase in write throughput for all platforms as compared to write-through caching. Before discussing the effect of write-back caching on latency for individual devices and for Parity RAID arrays, the paper describes how the write-back cache firmware was designed and tied directly to Parity RAID firmware.

The features chosen for write-back caching were extensively benchmarked against data integrity issues. The addition of settable timers allows customers to flush write data destined for devices that are idle for a specific length of time. To reduce the number of read/modify/writes required to update parity on small write operations, designers tied flush algorithms to RAID. Flush algorithms for write-back caching are vital to customer data integrity and to latency reduction. The flush algorithms actually allow Parity RAID to simulate RAID level 3 operations because of the nonvolatile write-back cache.

As mentioned earlier, Parity RAID configurations suffer a penalty on small write operations that includes a series of read and write operations and XOR operations on blocks of data to update RAID parity. The write-back cache firmware was designed with specific attributes to enhance Parity RAID write operations in general, and not just to enhance small write operations. The designers intentionally chose to overcome both the small-write penalty and the inherent lack of high bandwidth that Parity RAID delivers.

Table 2 StorageWorks Controller Write Request Throughput with Write-back Caching

Controller	Write Requests per Second
HSJ30/HSJ40	1,350
HSD30	900
HSZ40	1,850

The nonvolatile write-back cache module afforded the firmware designers more choices for Parity RAID write request processing and data flush algorithms. The designers pursued techniques to speed up all write operations by performing write aggregations (i.e., combining data from multiple write requests and read cache data) in three dimensions:

1. Contiguous aggregation, in which the firmware looks for consecutive block requests and ties them together into one device request, thus eliminating separate device requests.

2. Vertical aggregation, in which the firmware can detect two write operations to the same block, thus eliminating one write operation.

3. Horizontal aggregation (for Parity RAID operations only). This type of aggregation occurs when all data blocks within a Parity RAID strip are present in the write-back cache. In such cases, the firmware can write to all RAID set members at once, in combination with the FX chip (discussed later in this section) on-the-fly hardware XOR operations during the RAID set member writes. The original request can cause horizontal aggregation to take place if all blocks within a strip are part of the first write request. The firmware can also perform horizontal aggregation after processing several write requests. In this way, the parity write operation directly follows the data write operations. Horizontal write aggregation potentially cuts physical device access in half when compared to normal RAID write operations that require data members to be read.[2,8] The result is pseudo-RAID level 3 operation, because the write-back cache is combined with the horizontal aggregation cache policy.

The performance gain for individual disks and for Parity RAID arrays from using write-back caching is dramatic, resulting in higher write throughput and low latency. The write-back cache actually smoothes out differences in performance that are typical of workloads that have different read/write ratios, whether or not Parity RAID is utilized.

Figure 8 shows the relative latency for a controller with and without write-back caching enabled. The configurations tested comprised individual devices

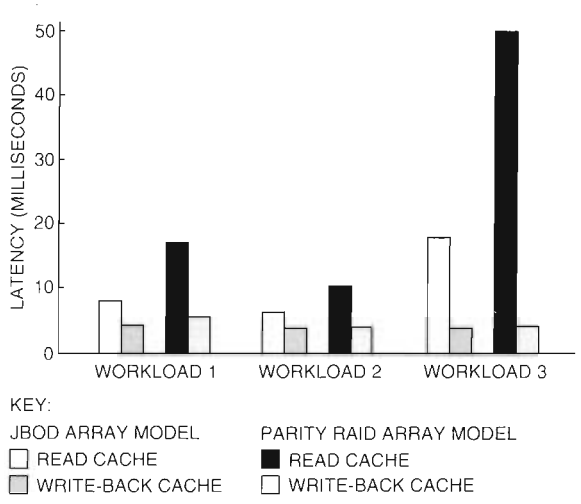


Figure 8 HSJ40 Array Latency Comparisons

and Parity RAID units (in a five-plus-one configuration). The performance measurements were taken from a version 2.0 HSJ40 array controller.

Workload 1 has a read/write ratio of 70/30, i.e., 70 percent of the requests were read requests and 30 percent were write requests. Workload 2 has a read/write ratio of 84/16. Workload 3 has a ratio of 20/80. In all workloads, the latency for individual devices and for Parity RAID units is lower when write-back caching is enabled than when only read caching is enabled. In fact, when write operations dominate the I/O mix, latency for Parity RAID units is the same as for the workloads in which read operations are predominant!

RAID/Compare Hardware

StorageWorks controllers contain a hardware Parity RAID and data compare accelerator called FX, a gate array that performs on-the-fly XOR operations on data buffers. Parity RAID and data compare firmware use this gate array to accelerate Parity RAID parity calculations and host data compare requests. The FX chip is programmed to (1) observe the bus, (2) "snoop" the bus for specific addresses, (3) perform the XOR operation to compare the associated data on-the-fly with data in a private memory called XBUF memory, and (4) write the data back into the XBUF memory.

XOR operations can take place as data is moving from buffer or cache memory to device ports or vice versa. The FX can also perform direct memory access (DMA) operations to move the contents of buffer or cache memory to or from XBUF memory.

The designers determined that hardware acceleration of XOR operations for Parity RAID firmware would speed up RAID parity calculations and thus further improve Parity RAID latency and throughput. The firmware also supports FX compare operations, which eliminates the need for SCSI-2 devices that have implemented compare commands and for speeding up compare requests from hosts.

Common Hardware Platform

To produce a high-performance controller in all three performance dimensions—latency, throughput, and data transfer rate—the designers of StorageWorks controllers faced the challenge of creating a new controller architecture and using new technology. In addition, they had to do so at a reasonable cost.

Although each has its own specific host interface hardware, the CI, DSSI, and SCSI controller variants share a common hardware core. Commonality was desired to control the development costs and schedules for such large engineering projects. To deliver high performance and commonality, the designers investigated several controller architecture alternatives. The first architecture considered was similar to Digital's HSC50-95 controller, incorporating similar bus structures, processing elements, and memories, but newer technology. Figure 9 shows the HSC architecture.[9]

The HSC architecture is a true multiprocessor system. It contains a private memory for its policy processor, which manages the work that is coming from the host port interface and queues this work to the device interface modules. Data then flows between the host port and device modules to and from hosts. The modules have two interfaces (buses) for access to command processing and data movement. These buses are called the control memory interface and the data memory interface. The policy processor queues work to the host port and device modules through the control memory interface, and then the modules process the data over the data memory interface.

Using this architecture would have been too expensive. The controller cost had to be competitive with other products in the industry, most of which currently cost considerably less than the HSC controller. The HSC bus architecture required three different memory interfaces, which would require three different, potentially large memories. The designers had to pursue other options that met the cost goals but did not significantly reduce

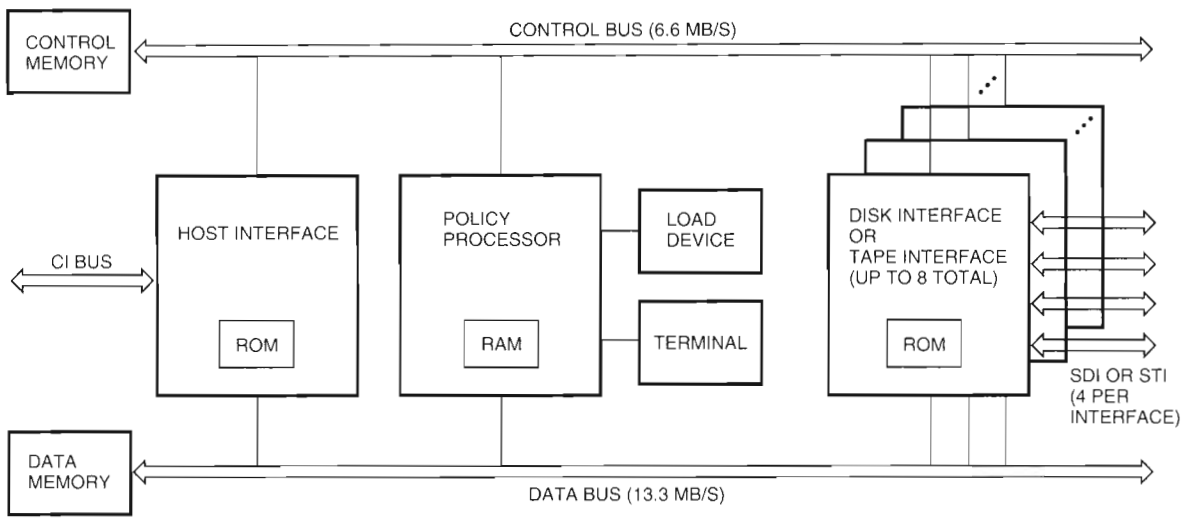


Figure 9 Block Diagram of the HSC Architecture

performance. They considered single internal bus architectures, but during simulation, these options were unable to meet either the initial or the long-term cost goals.

Figure 10 shows the controller architecture option that became the common hardware base for StorageWorks controllers. This architecture contains three buses and two memories. A third small memory is used for Parity RAID and data compare operations but does not drastically increase controller cost. The architectural design allows the policy processor to access one memory while a device or host port processor accesses the other memory.

The architecture achieves a lower overall cost than the HSC architecture yet achieves similar performance. The new architecture, with fewer memories, does not significantly reduce the performance, while the newer technology chosen to implement the controller enhances performance. The bus bandwidth of the new controller is much higher than that of the HSC controller. Consequently, a more cost-effective solution that uses

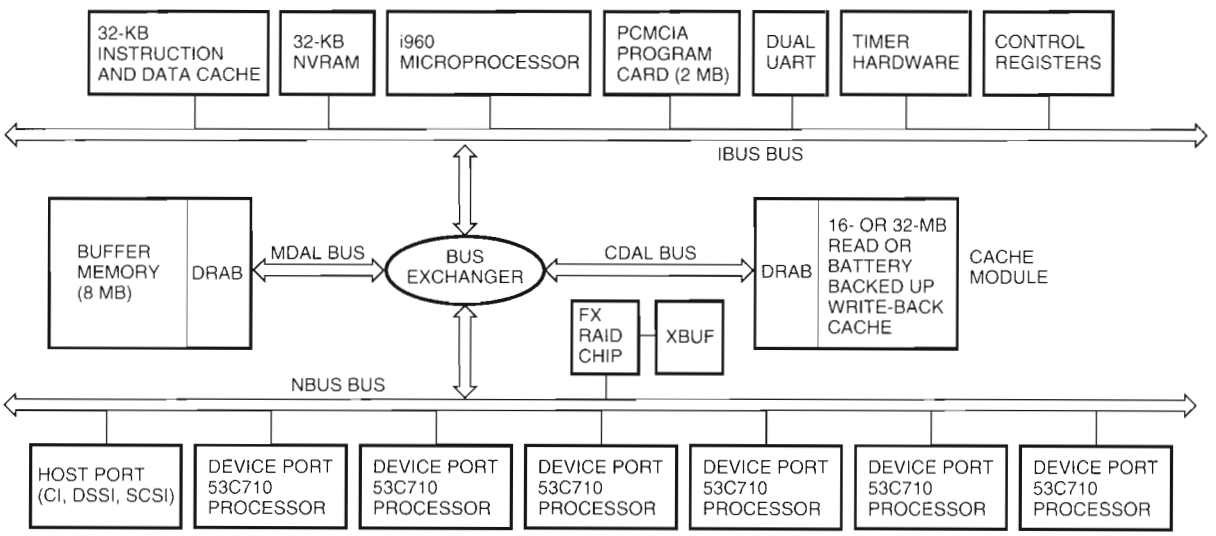


Figure 10 HSx40 Controller Architecture

a less-costly architecture can attain similar to better performance.

The extreme integration of hardware to the very large-scale integration (VLSI) level allowed for a much smaller enclosure than that of the HSC controller, even with a dual-redundant controller configuration (see Figure 3). A StorageWorks dual-controller configuration measures 56.5 by 20.9 by 43.2 centimeters (22 by 8 by 17 inches), which is approximately one-tenth the size of the HSC controller.

Common Controller Platform The common controller platform consists of the controller without the associated host port. The common core of hardware consists of the policy processor hardware, the SCSI-2 device port hardware, and the cache module. The controller-specific host port interface hardware includes either the CI, the DSSI, or the SCSI interface.

Policy Processor Hardware The StorageWorks controller policy processor is Intel's 25-MHz i960CA microprocessor, which contains an internal instruction cache and is augmented by a secondary cache external to the processor. The secondary cache relieves the potential bottleneck created by shared memory between the policy processor and host/device port processors.

The designers had to make trade-offs in two areas: the memory speed/cost and the number of buses. After simulation, the external instruction and data cache showed a significant performance improvement, given the chosen shared-memory architecture. The cache covers the first 2 MB of buffer memory, where policy processor instructions and local processor data structures reside and where most of the performance gain for the policy processor would be achieved.

The policy processor uses the IBUS exclusively to fetch instructions and to access the program storage card, the NVRAM, the DUART, and the timers.

Program Storage StorageWorks firmware is contained on a removable program card for quick code upgrades and to eliminate the need for a boot read-only memory (ROM) on the controller. The program card is a PCMCIA, 2-MB flash electrically erasable, programmable, read-only memory (EEPROM) card that contains the firmware image. Designers chose the PCMCIA card to facilitate code updates in the field, where host-based downline loading of firmware was not supported. Although the PCMCIA card cost more than EEPROM chips attached to the module, the designers felt that the benefits of such a design outweighed the additional cost.

On each initialization, the controller reads the firmware image on the program card and copies the image to the shared memory. The firmware executes from the shared buffer memory.

Dual UART (DUART) The DUART is used for two reasons:

1. Maintenance terminal connection. The maintenance terminal is a means of entering controller system management commands (with the command line interpreter, which is the user interface for controller configuration management) and is also a status and error reporting interface. Designers made extensive use of this interface for debugging controller hardware and firmware. Use of the maintenance terminal connection is optional. The interface remains on the controller so that users can direct controller management and status reporting, if desired.
2. Failover communication between two controllers in a dual-redundant configuration. The communication path is used to share configuration and status information between the controllers.

Shared Buffer and Cache Memory The dynamic random-access memory (DRAM) buffer (or shared memory) has at its heart the dynamic RAM and arbitration (DRAB) chip. This chip supports the buffer and cache memory accesses from the policy processor and from the host and device ports. The data transfer rate supported by the shared memory is approximately 35 megabytes per second (MB/s).

The DRAB chip contains error-correcting code (ECC) hardware to correct single-bit memory, to detect multibit errors, and to check and generate bus parity. This feature allows the controller to survive partial memory failures, which was a fault-tolerant goal for the controller.

The decision to use DRAM chips in the memory design rather than static random-access memory (SRAM) chips led to the use of ECC. DRAMs were chosen because of their cost and power savings over equivalent SRAM. However, because the designers expected large amounts of DRAM (as much as 40 MB) to be present on a controller and its associated cache module, the statistical error probabilities were high enough to warrant the use of ECC on the memory. The combination of DRAM and ECC was less costly than an equivalent amount of

more reliable SRAM. The use of parity on the buses is a standard feature in all StorageWorks controllers. The bus parity feature provides further error detection capability outside the bounds of the memory because it covers the path from memory to or from external host or device interfaces.

The DRAB chip also controls access to the cache module in conjunction with slave DRAB chips on the cache module associated with the controller. These DRAB chips provide refresh signals for the DRAM buffer or cache memory that they control; whereas, the master DRAB on the controller module provides arbitration for cache accesses that originate from the various sources on the controller module. Slave DRAB chips can also be accessed by the dual-redundant partner controller, depending on the two controller LOCK signal states.

The controller firmware uses 8 MB of shared buffer memory to execute the program image, to hold the firmware data structures, and to read and write-through cache data (if no cache module is present). The i960CA policy processor and the host and device data processing elements on the NBUS can all access buffer memory.

Cache Memory Each cache memory module contains one slave DRAB chip and 16 or 32 MB of DRAM, and also two ports into the module (one from each controller) for use in failover. Each cache module optionally contains batteries to supply power to the DRAM chips in the event of power failure for write-back caching and Parity RAID use. The cache modules are interchangeable between controller types.

Parity RAID XOR and Compare Hardware The Parity RAID XOR and compare hardware consists of the FX gate array and 256 kilobytes (KB) of fast SRAM. The FX allows concurrent access by SCSI-2 device port hardware and the policy processor. The FX compares the XOR of a data buffer (512 bytes of data) that is entering or exiting an attached device with the XOR buffers in the fast SRAM. The policy processor uses the FX to perform compare operations at the request of a host and perform DMA operations to move data to and from memories. This hardware is common across all the controller platforms for Parity RAID and compare firmware.

SCSI-2 Device Port Hardware The device ports (three or six, depending on the controller model) are controlled by Symbios Logic (the former NCR Microelectronic Products Division of AT&T Global Information Solutions Company) 53C710 SCSI-2 processor chips. The SCSI-2 processor chips reside on the NBUS and access the shared-memory cache for data structure and data buffer access. These processors receive their work from data structures in buffer memory and perform commands on their specific SCSI-2 bus for read or write operations.

The Symbios Logic chip provided the most processing power, when compared to the other chips available when the controllers were designed. The designers felt that direct control of SCSI-2 interfaces by the policy processor or a separate processor was too costly in terms of processor utilization and capital expense. The Symbios Logic chips do require some policy processor utilization, but the designers considered this acceptable because high-performance architectural features in the policy processor hardware compensated for the extra processor utilization.

The SCSI-2 device port supports the SCSI fast, single-ended, 8-bit interface.[1] The data transfer rate supported by this interface is 10 MB/s.

Host Port Hardware The host port hardware is either a CI, a DSSI, or a SCSI interface implemented with gate arrays or Symbios Logic 53C720 SCSI-2 processors. The host port hardware, the only noncommon hardware on a StorageWorks controller, requires a separate platform to support each host interface.

The CI interface is made up of a gate array and CI interface hardware that performs DMA write or read operations from shared memory or cache memory over the NBUS. The maximum data transfer rate supported by the CI hardware is approximately 8 MB/s.

The DSSI interface utilizes a Symbios Logic 53C720 chip coupled with a gate array and DSSI drivers to receive and transmit data to or from the DSSI bus. The DSSI interface is 8 bits wide, and the maximum data transfer rate supported by the DSSI hardware is 4.5 MB/s.

The SCSI interface also uses a Symbios Logic 53C720 chip coupled with differential drivers to provide a SCSI-2, fast-wide (i.e., 16-bit) differential interface to hosts. The maximum data transfer rate supported by the SCSI-2 interface is 20 MB/s for fast-wide operations.

Table 3 shows the current (version 2.0) maximum measured (at the host) data transfer rate performance numbers for StorageWorks controllers.

Table 3 SCSI-2 Host Interface Performance

Controller	Read Data Transfer Rate (Megabytes per Second)	Write Data Transfer Rate (Megabytes per Second)
HSJ30/HSJ40*	6.7	4.4
HSD30	3.2	2.8
HSZ40**	14	8.0

* In a multihost environment

** Measured for the HSZ40-B controller

Summary

The StorageWorks HS-series array controllers were designed to meet the storage subsystem needs of both Digital and non-Digital systems, thereby entering the world of open systems. The architecture for the HSJ30, HSJ40, HSD30, and HSZ40 controllers has achieved the initial project goals and provides

1. Open systems capability. A SCSI-2 device interface allows many types of disk, tape, and optical devices to be attached to the HSJ30, HSJ40, and HSD30 controllers. The HSZ40 controller, which is currently a disk-only controller, provides a SCSI-2 host interface that allows the controller to be attached to Digital and non-Digital computers.

2. High availability. Controller fault tolerance and RAID firmware yielded a highly available StorageWorks storage subsystem.

   The dual-redundant controller configuration allows each of a pair of active controllers to operate independently with host systems, while sharing device ports, configuration information, and status. This design allows both controllers to achieve maximum performance. The dual-redundant configuration also provides fault tolerance if one controller fails, because the surviving controller serves the failed controller's devices to the host computers. The dual-controller configuration, combined with StorageWorks controller packaging, results in a highly available controller configuration with built-in fault tolerance, error recovery, and battery backup features.

   Parity RAID controller firmware, combined with StorageWorks device packaging, allows for highly available disk configurations that are less costly than mirrored configurations. Furthermore, Parity RAID firmware performs automatic Parity RAID management and error recovery functions in the event of a failure and utilizes spare device pools in conjunction with user-defined Parity RAID configuration management policies. The StorageWorks Parity RAID implementation exceeds the requirements of the RAID Advisory Board for RAID availability features.

3. High performance. The HSJ30/HSJ40, HSD30, and HSZ40 controllers achieved the respective initial performance goals of 1,100, 800, and 1,400 I/Os per second. The controllers met the low request latency goals by streamlining firmware where possible and by introducing write-back caching. Write-back caching firmware dramatically reduces latency on all write requests, and write-back cache hardware provides battery backup for data integrity across power failures. Furthermore, the write-back cache overcomes the RAID level 5 small-write penalty and high data transfer rate inefficiencies and thus provides high performance with Parity RAID disk configurations. StorageWorks Parity RAID firmware implements many of the RAID Advisory Board optional performance features to produce a high-performance RAID solution.

   A common controller processing core was successfully developed for the HSJ30/HSJ40, HSD30, and HSZ40 controllers. More than 85 percent of the firmware is common to all three controller platforms, which allows for ease of maintenance and for the same look and feel for customers. The architecture and the technology used resulted in a core controller design that supports a high data transfer rate for all StorageWorks controller platforms.

These achievements represent the large engineering investment that Digital has made to move into the open systems market with new technology for its storage solutions. These controller platforms are the basis for future controller architectures and

platforms that utilize the knowledge and experience acquired during the development of the StorageWorks HS-series array controllers.

Acknowledgments

The StorageWorks array controller project was the cooperative effort of a large number of engineers who sacrificed considerable personal time to achieve the project goals. The following people and groups contributed to the success of the product: Bob Blackledge, Diana Shen, Don Anders, Richard Woerner, Ellen Lary, Jim Pherson, Richard Brame, Jim Jackson, Ron McLean, Bob Ellis, Clark Lubbers, Susan Elkington, Wayne Umland, Bruce Sardeson, Randy Marks, Randy Roberson, Diane Edmonds, Roger Oakey, Rod Lilak, Randy Fuller, Joe Keith, Mary Ruden, Mike Richard, Tom Lawlor, Jim Pulsipher, Jim Vagais, Ray Massie, Dan Watt, Jesse Yandell, Jim Zahrobsky, Mike Walker, Tom Fava, Jerry Vanderwaall, Dave Mozey, Brian Schow, Mark Lyon, Bob Pemberton, Mike Leavitt, Brenda Lieber, Mark Lewis, Reuben Martinez, John Panneton, Jerry Lucas, Richie Lary, Dave Clark, Brad Morgan, Ken Bates, Paul Massiglia, Tom Adams, Jill Gramlich, Leslie Rivera, Dave Dyer, Joe Krantz, Kelly Tappan, Charlie Zullo, Keith Woestehoff, Rachel Zhou, Kathy Meinzer, and Laura Hagar. Thanks to the CAD team, the StorageWorks packaging and manufacturing team, the software verification team, and the problem management team. A final thanks to our OpenVMS and DEC OSF/1 operating system partners and to the corporate test groups, all of whom worked with our engineering team to ensure interoperability between the operating systems and the controllers.

References and Notes

1. *Information Systems—Small Computer Systems Interface-2 (SCSI-2),* ANSI X1.131-1994 (New York: American National Standards Institute, 1994).

2. D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," Report No. UCB/CSD 87/391 (Berkeley: University of California, December 1987).

3. The RAID level 5 small-write penalty results when a small write operation does not write all the blocks associated with a parity block. This situation requires disk reads to recalculate parity that must then be written back to the RAID level 5 unit to achieve data redundancy.

4. P. Massiglia, ed., *The RAIDbook: A Source Book for Disk Array Technology,* 4th ed. (St. Peter, Minnesota: The RAID Advisory Board, September 1994).

5. P. Biswas, K. Ramakrishnan, D. Towsley, and C. Krishna, "Performance Analysis of Distributed File Systems with Non-Volatile Caches," *ACM Sigmetrics* (1993).

6. Parity RAID unit reconstruction of data and parity from a failed array member means regenerating the data block-by-block from the remaining array members (see Figure 6) and writing the regenerated data onto a spare disk. Reconstruction restores data redundancy in a Parity RAID unit.

7. Metadata is information written in a reserved area of a disk. The information, which takes up approximately 0.01 percent of the total disk capacity, describes the disk's configuration and state with respect to its use in a Parity RAID unit.

8. P. Biswas and K. Ramakrishnan, "Trace Driven Analysis of Write Caching Policies for Disks," *ACM Sigmetrics* (1993).

9. R. Lary and R. Bean, "The Hierarchical Storage Controller, A Tightly Coupled Multiprocessor as Storage Server," *Digital Technical Journal,* vol. 1, no. 8 (February 1989): 8–24.

Christoph J. Bußler |

Policy Resolution in Workflow Management Systems

One crucial function of a workflow management system (WFMS) is to assign tasks to users who are eligible to carry them out. Except in simple workflow scenarios, roles such as secretary and manager are not a sufficient basis for determining eligibility. Additionally, WFMSs are deployed not only in group settings by small companies but also worldwide by large enterprises. Since local laws and business policies have to be followed, task assignment policies for the same task generally differ from country to country and, therefore, must be specified locally. The Policy Resolution Architecture (PRA) model provides more generality and expressiveness than role models do and at the same time supports the independent specification of task assignment policies in different parts of an enterprise. PRA can be used to model arbitrary organization structures and to define realistic task assignment (eligibility) rules by means of precisely defined organizational policies. Thus, PRA provides real-world organizations with a precise, simple means of expressing their complex task assignment policies.

A workflow management system (WFMS) is a software system that manages the flow of work between participants or users according to formal specifications of business processes called workflows. A workflow specifies tasks to be performed and their execution order. Additionally, a workflow specification defines the internal flow of data between tasks as well as all applications required to carry out the tasks. For example, a travel expense reimbursement workflow specifies the tasks of filling, checking and signing a form, and reimbursing an amount. This workflow specifies that the form must be signed before an amount is reimbursed. The workflow specification also defines the flow of the expense form between tasks and the required spreadsheet application. Finally, for each task of a workflow, some rule has to be in place that specifies the users who are eligible to carry out the task. This set of eligible users is determined at run time, and the task is subsequently assigned to them.

One of the key issues in successfully deploying WFMSs in an enterprise is the correct assignment of a given task to eligible users. An eligible user is one who is capable of and responsible for carrying out an assigned task. This distinction is important because not every user who is capable of performing a task is necessarily responsible for it. The successful completion of a task, however, often requires that crucial, irreversible decisions be made by a person who is responsible for the task. Making the right decisions and then carefully and responsibly carrying out the task is essential to conducting business successfully.

The criteria used to determine an eligible user for a task are manifold. A user must have a specific set of capabilities to be able to carry out the task. Additionally, the position of a user in the organization hierarchy and/or the reporting structure of the organization can determine if the user is responsible for the task. Furthermore, limits placed on a user's decision-making authority can affect eligibility. For example, not every salesperson is authorized to accept an order that leads to a significant increase in manufacturing output. Such an order requires special attention and internal coordination by a senior sales representative. When cost-optimized task assignments are made, the experience of the user as well as the user's skill set has to be taken into consideration. Highly experienced users are in most cases expensive resources, but usually they can complete tasks faster than users with average experience. Although users with either level of experience may have sufficient experience to carry out a specific task, if deadlines

are involved or extreme caution with respect to quality is necessary, a highly experienced user might be appropriate. In such cases, the additional cost would be justified.

The previous discussion demonstrates the necessity of a precise definition of eligible users for a given task. Such a definition, i.e., set of task assignment rules, should contain all the criteria used to determine eligible users for the task. Early in the development of Digital's ObjectFlow WFMS product, the concept of roles was considered sufficient to model the assignment of tasks to users.[1] However, an analysis of distributed enterprise-wide production workflows clearly showed that using roles as the only assignment mechanism has limited value in determining eligibility.[2] The need for a far more expressive, general, and flexible approach became obvious. The analysis also revealed that workflows are often reused in different parts of an enterprise. A prominent example is the travel expense reimbursement workflow, which is discussed throughout this paper. Although a workflow is reused, however, the task assignment policies may differ greatly in the various parts of an enterprise. This difference is due to the need to adhere to local laws and/or to business-related deviations from the general rules.

Based on the requirements derived from several case studies of complex workflows, the Policy Resolution Architecture (PRA) was developed to provide a comprehensive way of specifying task assignment rules.[2] To support the fact that different parts of an organization may require different assignment rules, PRA and its implementation were designed as separate components. PRA incorporates three major elements and thus provides

- Concepts that enable the modeling of any organization structure (not just roles and groups) without prescribing structures that are application dependent.
- Task assignment rules as entities in themselves, separate from a workflow specification. This makes it possible for each of the different parts of an enterprise to have its own set of task assignment rules for the same workflow.
- A language that enables the explicit specification of organization schemas and task assignment rules. Specifications are processed by a component called the policy resolution engine during workflow execution.

Before explaining PRA in detail and providing the rationale for its development, the paper introduces the key concepts of workflow management. This introduction presents a seemingly simple workflow that specifies travel expense reimbursement, which is later used to introduce the design objectives of PRA. Note that a real travel expense reimbursement workflow for production is by far more complex than the example used in this paper. A large distributed enterprise endeavors to reuse the same workflow in all of its parts because reuse facilitates administration and leverages the development investment. At the same time, such an enterprise probably sponsors numerous business trips, which makes the travel expense reimbursement workflow an excellent candidate to use as an example.

Workflow Management

This section introduces a model of workflow management. The discussion begins with a survey of preliminary work. The survey suggests the motivation for workflow management and enumerates some areas in which workflow management is deployed. The key concepts of the workflow model are then used to model a workflow example, i.e., the travel expense reimbursement workflow. The section concludes with a definition of workflow management systems.

Historical Survey

Looking back in history reveals that workflow management has many roots. The most important are office automation, software process management, manufacturing, and transaction processing. The following short survey of achieved results is given to help the reader understand the motivation for workflow management. The discussion also explains the choice of workflow management concepts. The list of previous and related works indicates the range of literature that exists.

Office Automation One of the primary roots of workflow management is undoubtedly office automation. Early research led to the development of models and tools to support office workers.[3-9] What emerged were not only desktop applications that imitate concepts such as in basket, out basket, forms, and documents but also models of the procedures that the office workers follow while doing their jobs.[10,11] Furthermore, systems were developed that execute the office procedures to actively manage the flow of work within offices.[12,13]

Software Process Modeling A second major root of workflow management is software process modeling and execution.[14-25] The focus of research in this area is the automated support of software development processes. Concepts comprise process models like the waterfall model or the spiral model, deliverable code, installation and operation manuals, requirements documents, and test cases.[26,27]

Manufacturing Traditionally, formalized procedures that are executed repeatedly are inherent to manufacturing, another root of workflow management. Manufacturing involves not only production processes but also preproduction procedures starting from, for example, the release of computer-aided design (CAD) drawings to the preparation of shop floor schedules.[28-31]

Transaction Processing Another important area that influenced the development of workflow management is transaction processing. After the concept of atomicity, consistency, isolation, and durability (ACID) transactions was developed, researchers proposed more advanced transaction models for processing several interdependent tasks that must be transactional and recoverable.[32-39]

Coordination Theory, Enterprise Modeling, and Speech Act Theory Another area of research that contributed to the idea of workflow management is coordination theory.[40,41] This area looks at processes as one form of coordination and tries to apply interdisciplinary research results to it. The research area of enterprise modeling focuses on the modeling of the whole multifaceted enterprise.[42-49] Enterprise activities are one part of an enterprise that drives the enterprise processes. The speech act theory is an attempt to model the conversation between humans.[50] Some research follows the direction that a workflow is an interwoven chain of speech acts.[51]

Early Application-independent Approaches In addition to the application-specific roots of workflow management, early approaches that modeled processes independent of application areas provided motivation for workflow management.[52-54]

The term process appears in all the areas of work mentioned above. Also, all these research areas deal with data, e.g., documents, CAD drawings, and orders. Most approaches have some notion of subject or agent. The question arose among researchers, Does each area need its own definition of terms, modeling language, and execution mechanism, or is it possible to provide general concepts that need to be customized only for a specific area of application? This question triggered the development of the concept of workflow, whose goal it is to serve as the general and customizable concept.

Workflow Management Concepts

After the specific application semantics (e.g., documents, office workers, release procedures, and CAD drawings) have been abstracted, the basic concepts of workflow management can be distilled from the various approaches mentioned above. Although workflow management is independent of specific application semantics, it does support all the application areas cited. It provides an integrated set of underlying concepts that can be customized to model the semantics of each application area. Workflow management is analogous to relational database systems. Such systems know how to model and implement tables and how to process queries; however, they do not know about the specific concepts of an application area that are implemented by user-defined tables, e.g., addresses and orders.

The following list introduces the basic concepts of workflow management by enumerating the major aspects that make up a workflow specification:[14]

- Functional aspect. The functional aspect describes what has to be done, without saying how, by whom, and with which data. The functional aspect provides two concepts: elementary workflows and composite workflows. Elementary workflows are tasks that can be carried out by one person, program, or machine. For brevity, elementary workflows are called steps. Composite workflows bundle either elementary workflows or other composite workflows to higher-level tasks. In this way, a reuse hierarchy is built, since the bundled workflows may very well stand by themselves. Generally, these higher-level tasks can no longer be achieved by a single person, program, or machine but require several such entities. A workflow that bundles other workflows references them. As a naming convention, a workflow that is referenced by some other workflow is called a subworkflow. The referencing workflow is called the superworkflow. The topmost workflow of a reuse hierarchy is called the top-level workflow.

- Behavioral aspect. The behavioral aspect describes the execution order of the subworkflows of a workflow. Constructs that describe the order include sequence, conditional branching, parallel branching, and the looping and/or joining of parallel or conditional execution paths.
- Informational aspect. The informational aspect is twofold: first, it describes the local variables of a workflow and the external data referenced; second, it describes the flow of data from subworkflow to subworkflow.
- Organizational aspect. The organizational aspect describes who is eligible to carry out a step. The "who" can be a human (e.g., an office worker), a program (e.g., a compiler in a software process), or a machine (e.g., a cell in a shop floor). The term user was chosen to represent all three. Most available WFMSs offer the concept of roles to model the organizational aspect. A role usually groups a set of users. At run time, tasks are assigned to roles and all users grouped by these roles are assigned the task. Although this method of task assignment is adequate for certain workflows such as departmental workflows, as shown later in the section Task Assignment in a Travel Expense Reimbursement Workflow, roles are not sufficient to handle workflows that are deployed in an enterprise-wide or international setting.

The literature discusses additional aspects, e.g., a historical aspect and a technological aspect.[55] The historical aspect is used to specify the kind of information to be stored in a historical database during the execution of a workflow, e.g., starting times or values of variables. Instead of having the default strategy of saving all data, the workflow specifies in the historical aspect only the important data that must be stored. The technological aspect allows the definition of which application program or programs are available to carry out a step. At run time, these application programs are made available to the user. In principle, it is not possible to enumerate all necessary aspects completely in advance. Depending on the application area to be modeled, additional aspects might appear and require support.

The paper now shows how the key concepts of workflow management can be applied, i.e., customized, to model a specific workflow type. The example used is a sample travel expense reimbursement workflow.

Travel Expense Reimbursement Workflow

Figure 1 shows the graphical representation of a simplified workflow for the reimbursement of travel expenses. (Examples of workflow language can be found in the literature.[55,56]) The workflow consists of four steps: (1) fill, (2) check, (3) sign, and (4) reimburse. The graphical representation shows the functional aspect (task structure) as ovals and the behavioral aspect (control flow) as solid arrows. The informational aspect (data flow) is displayed as forms; dotted arrows indicate the direction of the flow of data. The organizational aspect is omitted since the paper will focus later on this topic. The technological aspect is represented by icons of the software applications that are available to carry out the steps. The historical aspect is represented by icons that symbolize logs in which information must be recorded.

Step 1 of the travel expense reimbursement workflow, the fill step, enables a user to enter the relevant expenses incurred during a business trip into an electronic travel expense form. After a user has finished entering the data, validation must take place. The check step enables a user to look at the contents of the travel expense form. This user is prompted to validate the contents but cannot change entries. If the user who checks the form detects an error, the form is sent back to the user who initially filled it out, with a note that explains the reason for rejection. Otherwise, the form is forwarded to the next user who has to sign the form to approve the amount. After the sign step is complete, the amount can be reimbursed. The last step, reimburse, enables a user to add the amount spent to the next paycheck of the user who requested reimbursement.

This sample workflow is intentionally kept simple because beginning with the next section, the paper focuses solely on task assignment rules. In a real organizational setting, the workflow would involve more steps and additional execution paths. For example, a user who has to sign the form might detect an error. In this case, as in the check step, the form would be sent back to the user who initially filled it out.

Workflow Management Systems

Managing the flow of work among users is done by a software system called a workflow management system (WFMS). A WFMS contains all the specifications of the workflow types (e.g., a travel expense

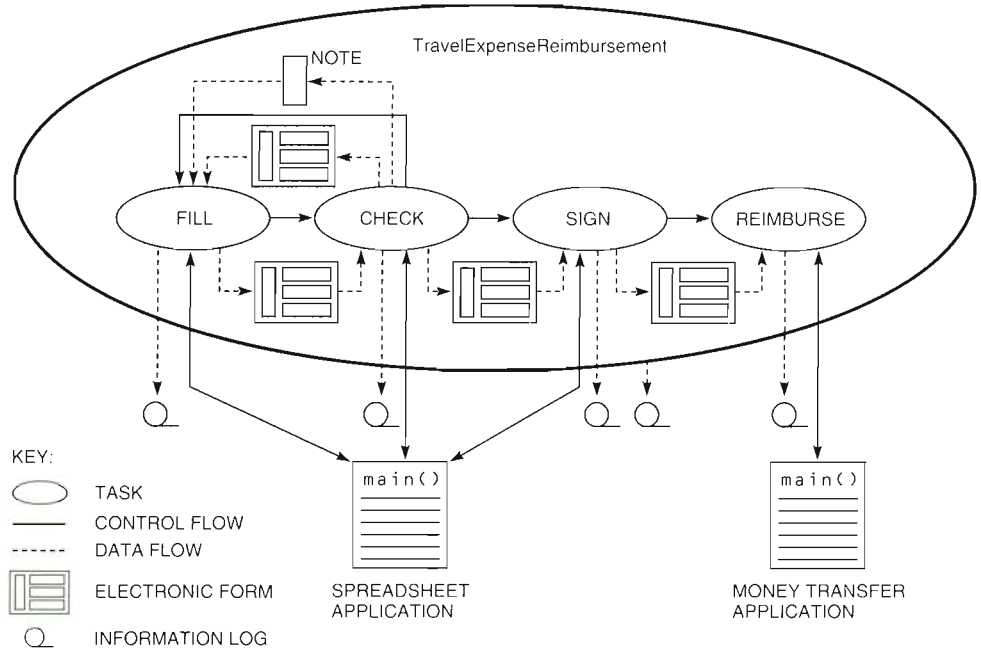


Figure 1 Travel Expense Reimbursement Workflow

reimbursement or a capital equipment order) that are modeled and released for production. If a user issues a request to start a workflow (e.g., if, after a business trip, a traveler starts a travel expense reimbursement workflow), the WFMS creates an instance of the requested workflow type. Of course, more than one instance of the same workflow type can exist simultaneously. A WFMS assigns the steps of a workflow to users according to the specified order of the behavioral, functional, and organizational aspects.

In general, a WFMS performs the following actions to execute a workflow instance:

- Determine the next steps to be executed.
- Determine the eligible users for these steps.
- Assign steps to eligible users.
- Wait for the result of each step.
- Transfer the result back to the step's superworkflow and record the step as complete.

The WFMS repeats these actions until all steps of a workflow are executed.[55,57-59] This list of actions has to be slightly modified if, in addition to steps, a workflow contains composite workflows in its list of subworkflows. In this case, the subworkflow is not assigned to users and the list of actions is applied to each of the subworkflows.

Each user who can potentially be involved in a workflow is connected to a WFMS by a private worklist, which is a graphical representation of a list of steps assigned to the user. Each entry in a user's worklist represents a task the user is eligible to carry out. A user can participate in more than one workflow at the same time. Normally, the user is free to choose from the worklist any item on which to start. In well-designed systems, the WFMS automatically starts the application programs that the user will require to accomplish the work. In this way, the user can begin work immediately.

Almost all prototype implementations or product developments allow the modeling of the four main aspects described previously. The list of workflow management systems is growing rapidly, and references to relevant literature are readily available.[57,57-61] References to literature that describes the deployment of workflow management systems in an application area are rare, however.[51,61,65-67]

The reminder of the paper focuses on the organizational aspect of workflow management. The paper discusses the derivation of the requirements that concepts of this aspect must meet and then introduces PRA as the model whose concepts

address the requirements. An analysis of the travel expense reimbursement workflow illustrates some of these requirements. Additional requirements are also described to provide a more complete set.

Task Assignment in a Travel Expense Reimbursement Workflow

The requirements that must be fulfilled by the concepts of the organizational aspect were derived from the travel expense reimbursement workflow example, the author's project work experiences, and Marshak's "Characteristics of a Workflow System—Mind Your P's and R's."[68] The following list describes task assignment rules for each step of the travel expense reimbursement workflow:

- Fill. The fill step can be executed by anyone in an organization who has the potential to travel. This assignment rule enables an employee to fill in a travel expense reimbursement form after a business trip. (An employee who did not travel can also fill in a form and claim expenses; however, the check and sign steps are intended to detect such misbehavior and to reject the form.) The user who fills in the form is referred to as the applicant and is known at run time.

- Check. The check step must be executed by a user who is able to play the role of secretary. To be able to validate the contents of the form, a user in this role is expected to know how a travel expense reimbursement form is structured and how to correctly fill in the form. This user is also expected to know the destination and the travel dates, and if the travel actually took place. Not all secretaries in an enterprise have this knowledge, but the secretary of the applicant's manager can be expected to know the information. This secretary usually plans the trip and often the meetings of the traveler. If the user who is able to play the role of secretary determines that the contents of the travel expense reimbursement form are sound, the form is forwarded to the next step; otherwise it is sent back to the applicant.

  The overall task assignment rule is therefore: Everyone who is able to play the role of secretary and reports to the same manager as the applicant is eligible to execute the check step. (Note that the term manager means a user who is able to play the role of manager.)

- Sign. The sign step has to be executed by a manager of the applicant because the manager normally has to approve spending by subordinates. Usually, there is only one user to whom the applicant reports and who is able to play the role of manager. If there are two such users, either can be responsible for signing the form and only one has to sign it.

  The overall task assignment rule is: Everyone who is able to play the role of manager and to whom the applicant reports is eligible to execute the sign step.

- Reimburse. The reimburse step must be executed by a financial clerk who is responsible for the group to which the applicant belongs.

  The overall task assignment rule is: Everyone who is able to play the role of financial clerk and who is responsible for the applicant's group is eligible to execute the reimburse step.

The requirements thus far derived from the example are

- Organization structure dependencies. To select one user relative to another (e.g., a user playing the role of secretary reporting to a user playing the role of manager) requires describing the users, the roles, and the dependencies (relationships). This description is called an organization structure. An organization structure contains all organizational object types like "user," "group," or "role," and the relationships among them like "reports to" or "supervises." Given such a structure, users can be selected based on their relationships to others. Users can also be selected based on attributes such as their absence status (i.e., whether they are on vacation or on a business trip) or their workload.

- Historical access. In some cases, the eligible user for a step cannot be determined locally, and historical information is required. For example, determining the user who can play the role of manager in one step might require knowing which user started the workflow. Therefore, it must be possible to query a log of the history of a workflow to derive the information necessary to make task assignments.

The following are additional requirements:

- Data dependency. In the travel expense reimbursement example used in this paper, the manager to whom the sign step is assigned can sign for any amount. In other cases, however, this

signatory power may have limitations. For instance, if the amount exceeds a certain value, a vice president and not the manager of the applicant must sign the travel expense reimbursement form. As this last example shows, task assignment may depend on data in the workflow.

- Delegation. A manager who is out of the office may want to delegate his/her tasks to keep business operations running smoothly. The appropriate task assignment rule would then have to be extended to incorporate the delegation of tasks. Depending on the status of the manager (e.g., on a business trip or on vacation), the work would be assigned to someone else (i.e., delegated). However, task assignment rules that incorporate delegation can be complex. Consider the situation in which a manager leaves on a business trip after work has already been assigned. In this situation (and also in the case where a manager has an excessive amount of work to accomplish), the manager must be able to dynamically delegate some or all of the already assigned tasks. Further consider that a manager may want to delegate different types of tasks not to the same user but to different users, depending on the type of task. To avoid leaking information or making an inexpedient assignment, the task assignment rule must make sure that the target users are eligible to receive the delegated task assignment.

- Separation of duty. Some scenarios require a separation of duty, i.e., two tasks must be performed by different users. For example, in the transfer of a large amount of money, two managers must sign the transfer form to double-check the transaction. Regarding the travel expense reimbursement workflow, a user who fills out the claim form should not also sign it. Task assignment rules must ensure that there is a separation of duty.

- Responsibility. As previously stated, a subworkflow can be either a step or a group of steps that may be a reuse of building blocks for larger workflows. A second use of a composite workflow is to explicitly express responsibility for workflows. Sometimes an application domain requires a user to take responsibility for a set of tasks even though the user does not actually execute the tasks. For example, consider a workflow that implements the start of a new product development. The investment plan depends on the development plan, which is based on a market analysis. A manager or a vice president is usually responsible for these three complex tasks (market analysis, development plan, investment plan) but not involved in the detailed work. In a WFMS, this situation would be modeled as a workflow called Product Development Start, which contains the three complex tasks as subworkflows. The Product Development Start workflow could then be assigned to a manager or a vice president to model responsibility. The assignment to this user means only that the user must acknowledge the start of the assigned workflow and therefore accept responsibility for it. The assignment does not imply that the user has to perform the detailed work. Thus, a WFMS must be able to assign not only steps to users but also composite workflows.

- Early/late allocation. Often, the application semantics clearly indicates the single user who should execute a task. In such cases, the related task assignment rule (e.g., the role of manager of applicant) passes to this user at run time. In other scenarios, however, successful execution of a task requires some capability that more than one user possesses. This capability is often expressed through a role (e.g., financial clerk, which is a role usually played by more than one user in large enterprises). In the single-user case, the task is assigned to that user regardless of the user's workload; this process is called early allocation. The user must carry out the task unless it is feasible to delegate it. In the multiple-user case, the task appears on the worklist of all users able to play the role. One user starts the task; in most cases, this user would not have the highest workload. Therefore, the final allocation of the task is made not by the WFMS but by the set of eligible users themselves. This process is called late allocation. In this case, if one user starts work on a step, the other users are no longer allowed to begin the task.[5,59] Subsequently, their assignment must be revoked. "Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems" describes a general mechanism for handling the revocation of assignments.[69]

The travel expense reimbursement workflow is used in the following discussion about the limitations of roles as a basis for task assignment rules. These limitations influenced the major design objectives of PRA, which are then discussed.

Roles As Task Assignment Rules

As stated earlier, roles have limited use as task assignment rules. Applying the role concept to the task assignment rules introduced above illustrates the limitations. Certainly, the term role has many definitions. In this paper, a role is an abstraction of a set of users. The abstraction criteria are the set of capabilities of a user. Whether or not a particular user belongs to the set of users abstracted by a role is defined by an explicit relationship between a user and a role called the "plays" relationship. A user who has a plays relationship with a role has the capabilities defined by that role, i.e., the user is able to play the role. For example, if both Ann and Joe are users who are able to play the role of clerk, then each one has the capabilities defined by this role and each is capable of executing the task. A user might have a wide range of capabilities and be able to play several roles at the same time. For example, a user might be able to play the role of employee and the role of manager simultaneously. Although this definition of role is not the only one, it is very common and often applied.[6,14,51,52,62,63,70,71]

For each task assignment rule that was introduced in the travel expense reimbursement example, a discussion follows about the extent to which roles support the requirements.

- Fill. The task assignment rule for the fill step is the only rule of the example that can be modeled completely with a role. Assume that every user is able to play the role of employee. If the fill step is assigned to the role of employee, every user can execute the step, thus modeling exactly the task assignment rule of the fill step.

- Check. Assigning the check step to the role of secretary does not model the full semantics of the desired task assignment rule. Such an assignment models only the requirement that a user has to be able to play the role of secretary to carry out the step. The assignment does not model the additional requirement that only those users who report to the same manager as the applicant are eligible.

- Sign. Analogous to the situation in the check step, assigning the sign step to the role of manager does not model that only a user to whom the applicant reports is eligible but that any manager is eligible.

- Reimburse. Assigning the reimburse step to the role of financial clerk ensures only that the step is assigned to a capable user. The assignment does not fulfill the additional requirement that this user must also be responsible for the group to which the applicant belongs.

The discussion of the last three task assignment rules demonstrates two tightly coupled limitations of using roles to model requirements.

1. The concept of roles cannot express organizational dependencies, such as relationships between users (e.g., "reports to" and "responsible for"). It only relates users to roles by a plays relationship. Furthermore, roles do not provide a means of introducing additional objects of organization structures like "group" and "department." The only two objects the concept of roles provides are "role" and "user."

2. The concept of roles, therefore, does not provide a sufficiently sophisticated language to express, for instance, that a user not only has to play a certain role but also has to relate to some other user in a particular way (e.g., "reports to" a particular user).

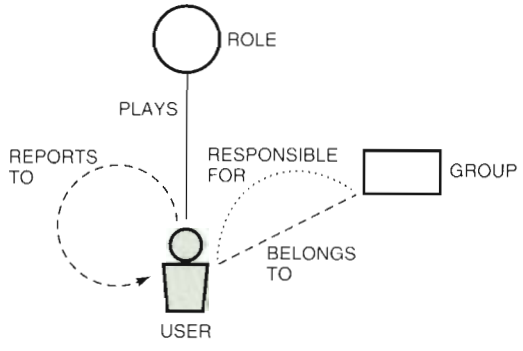
In addition, the other requirements like historical access, delegation, and separation of duty cannot be modeled at all using roles.

To overcome these limitations, PRA introduces the concepts of organization schema and organizational policy and the Policy Definition Language. A brief introduction follows. Details are presented in the section Policy Resolution Architecture.

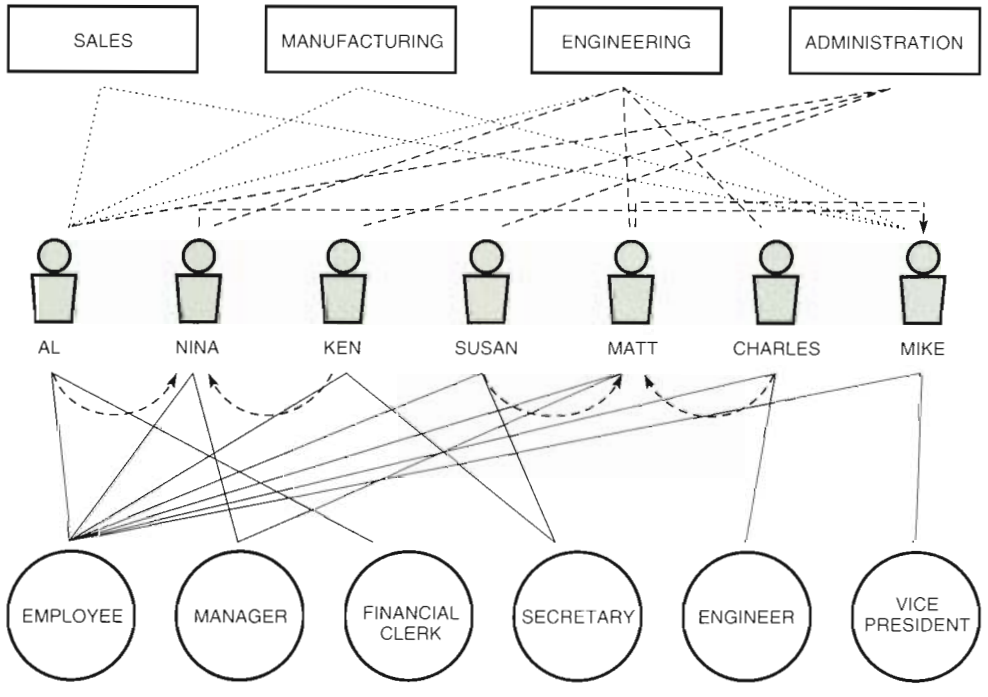
Organization Schema

One of the fundamental concepts of PRA is a freely definable organization schema. An organization schema contains all types of organizational objects and relationships that are available for modeling a particular organization. Figure 2a gives an example of an organization schema. If a defined schema is instantiated, it contains an organization structure. Since other objects besides roles are required to model an organization, relationships other than "plays" must be available. Some necessary additional relationships are "reports to," which relates two users, and "is responsible for" and "belongs to," which relate a user and a group. A freely definable organization schema, such as the one provided by PRA, allows designers to define roles as required by the workflow application.

Such a freely definable organization schema may seem to be a luxury, and a fixed organization



(a) Sample Organization Schema



(b) Sample Organization Structure

Figure 2 Sample Organization Schema and Organization Structure for the Travel Expense Reimbursement Example

schema that provides the most relevant objects and relationships may seem sufficient. An analysis of various organization structures in different enterprises clearly shows, however, that a single organization schema is not adequate for all situations in which WFMSs can be deployed. An enterprise that deploys a schema in which the semantics of the modeled objects are fixed has to follow the semantics completely. Consequently, such a schema does not meet enterprise-specific needs.

Figure 2a shows a graphical representation of a sample schema for the travel expense reimbursement example. Although this schema may appear general and an adequate alternative to an all-embracing schema, it does not contain required organizational objects such as task forces with

a limited life span, committees, and departments. Also, this sample schema does not consider objects or relationships necessary for modeling delegation and relocation of employees. Figure 2b displays a superficial organization structure, i.e., an instantiation of the schema. Objects like user and role are depicted as icons, and relationships are depicted as arcs and solid, dashed, and dotted lines between the icons.

Approaches that go beyond using roles as a basis for task assignment commonly provide organizational objects in addition to roles and users, usually group and/or department objects.[2,6,8,59,72] The literature contains evidence that the schemas and the task assignment rules are fixed and have to be used as they are. Additionally, these approaches do not separate the workflow from the workflow specification, which makes the reuse of a workflow in a different organizational setting very difficult.

Organizational Policies As Task Assignment Rules

A second fundamental PRA concept is that of an organizational policy, which up to this point has been called a task assignment rule. An organizational policy specifies all the eligible users for a task by stating the criteria a user must meet. These criteria can include a role or roles that a user has to be able to play and relationships that a user has to have with other users or groups.

Figure 3a shows an example of an informal organizational policy for the sign step. This organizational policy specifies that if the WFMS is to assign the sign step, it will assign the step to the manager of the applicant if the amount is less than $1,000. Otherwise, it will assign the step to the vice president responsible for the applicant's group. A more advanced rule would not fix the amount at $1,000 but would make this amount dependent on the authorization level of the manager, as illustrated in Figure 3b.

The Policy Definition Language is PRA's formal language for specifying organizational policies. Policies written in this language are precise and executable by an execution engine called the policy resolution engine. Each time the WFMS is about to assign a step, the system evaluates the corresponding organizational policy to determine the set of users who can execute the task.

Policy Resolution Architecture

WFMSs operate in global, open, and distributed environments and in group, department, enterprise, and multiple-enterprise settings. The enterprise-level deployment of workflows is possible only if the underlying concepts and systems are developed appropriately. PRA is therefore based on several design principles that ensure a general approach that supports enterprise-level deployment.

```
(a)

WORKFLOW TravelExpenseReimbursement
STEP     sign
CRITERIA IF amount < 1000
         THEN manager of applicant
         ELSE VP responsible for applicant's group
         ENDIF


(b)

WORKFLOW TravelExpenseReimbursement
STEP     sign
CRITERIA IF amount < authorization level of applicant's manager
         THEN manager of applicant
         ELSE VP responsible for applicant's group
         ENDIF
```

Figure 3 Informal Organizational Policies for the Sign Step of the Travel Expense Reimbursement Workflow

Design Principles

The PRA design principles are reusability, security, generality, dynamics, and distribution.

Reusability In the travel expense reimbursement example, the sign step was modeled to approve travel expenses. Other workflows, like capital equipment orders, can reuse the sign step for similar tasks, e.g., to approve an order. If an organizational policy were attached to the step type itself, this assignment rule would serve to determine eligible users independent of the workflow in which the step is reused. Viewed from an organizational perspective, however, the reuse of steps in different workflows requires several policies. For example, the signing of a travel expense reimbursement form is carried out by a manager of the applicant, whereas the signing of a capital equipment order for an amount that exceeds a certain value is carried out by an appropriate vice president. Therefore, the sign step in the context of a travel expense reimbursement workflow has an organizational policy that defines the manager of the applicant to be eligible, whereas the sign step in the context of the capital equipment order workflow has a different policy, one that defines an appropriate vice president as eligible for the task.

The observation that a policy for a step depends not only on the step itself but also on the workflow in which the step is reused led to the decision to make organizational policies objects in themselves, independent of a workflow specification. Organizational policies name not only the step in which they are used but also the surrounding workflow. The design of organizational policies for a step depends on the context in which the step is to be reused.

As mentioned earlier, making organizational policies independent objects allows different organization structures to reuse a workflow. To achieve such reuse, each organizational setting has its own set of organizational policies for the workflow to be reused. These organizational policies are tailored to the specific needs and circumstances of the organizational setting.

Organizational policies can themselves be reused. Different steps may require the same set of eligible users, and, therefore, one policy would be sufficient for more than one kind of step (e.g., sign and fill) or for more than one use of the same kind of step. For example, a manager signs not only travel expense forms but also capital equipment orders. In both workflows, the organizational policy that defines the manager of the applicant depends on the authorization level. Both workflows can reuse the sign step, as can be seen in the policy shown in Figure 4a. If the authorization level depends on the workflow, the policy changes to take into consideration the specific kind of workflow, as shown in Figure 4b.

```
(a)

WORKFLOW TravelExpenseReimbursement | CapitalEquipmentOrder
STEP     sign
CRITERIA IF amount < authorization level of applicant's manager
         THEN manager of applicant
         ELSE VP responsible for applicant's group
         ENDIF


(b)

WORKFLOW TravelExpenseReimbursement | CapitalEquipmentOrder
STEP     sign
CRITERIA IF amount < authorization level of applicant's
                     manager depending on workflow type
         THEN manager of applicant
         ELSE VP responsible for applicant's group
         ENDIF
```

Figure 4 Informal Organizational Policies Showing Reuse of the Sign Step

Security Because changing an organizational policy may affect daily business operations, all users should not be able to make changes at will. For example, a user (applicant) should not be able to approve his/her own travel request. Organizational policies are therefore objects that must be properly secured to prevent users from performing unauthorized tasks. The decision to design organizational policies as objects makes it easier to secure the policies, because security mechanisms such as access control lists (ACLs) can be applied directly to objects.[73]

Designers considered and rejected the alternative approach of securing the workflow specification and, consequently, the organizational policies included in the specification. Workflow types do have to be secured to prevent unauthorized changes; however, securing the workflow specification would allow those who are eligible to change the workflow type to also change the associated organizational policies. Such an all-encompassing security design inhibits the separation of duty between workflow designers who care about how a business process is implemented by a workflow and organization designers who care about the organization structure and the user capabilities and responsibilities. Protecting workflows independently of organizational policies allows users to modify a workflow without allowing them to modify organizational policies and thus gain or grant unauthorized eligibility. Similarly, organization schemas and organization structures must be secured independently to prevent users from changing roles or relationships to gain or grant unauthorized authority.

Generality Although several standard organization structures prevail—strong hierarchical, matrix-shaped, function-oriented, and networked—hybrid organization structures exist, which contain a myriad of anomalies and exceptions. Independent of their organization structure, most enterprises have business processes that are potential candidates for a WFMS implementation. A WFMS that claims to be able to implement business processes in all kinds of enterprises must therefore be able to support all possible organization structures. A fixed organization schema is inadequate for such a universal implementation capability. Consequently, PRA supports the modeling of arbitrary organization schemas and allows WFMSs to implement any organization that might exist.

Following this general approach, it is apparent that a fixed set of assignment rules is also inadequate. The PRA design hence provides a language that enables users to define task assignment rules (organizational policies) as required by the workflows of an enterprise.

Dynamics Organizations change for many reasons, e.g., employee numbers fluctuate, restructuring takes place, groups join or split because of new product strategies, etc. Business operations and therefore workflows, however, must continue uninterrupted. To do so, the organization structure and the organizational policies of a WFMS must change to reflect the changes in the real organization. The decision to separate workflows from organization structures and organizational policies enables users to change versions independently. For example, an organizational policy can change while a workflow that uses it is running. If the change takes place before the WFMS assigns the step to a user, the WFMS will use the new version of the organizational policy instead of the old version. Policy changes result in neither the shutting down of the WFMS nor the stopping and restarting (from the beginning) of the workflow. This independence allows WFMSs to deal with the dynamics of an organization and make correct task assignments while changes are taking place.

Distribution Not only are enterprises becoming more distributed, but they are also increasing their worldwide operation. Nations have different local laws and policies because they decide autonomously on these issues. A local subsidiary has to adhere to local law, even though it belongs to a company that operates worldwide. For example, U.S. companies have a position called vice president. A U.S. company may have the rule that contracts with external suppliers of manufacturing parts must be signed by the vice president of manufacturing. If the U.S. company has a German subsidiary, by German law, this subsidiary is a company in itself and must have a person called *Geschäftsführer* who is responsible for the operations of the company. If the subsidiary wants to enter into a contract with a supplier, German law requires the *Geschäftsführer* to sign the contract even though the U.S. corporate organizational policy requires the vice president of manufacturing to sign. Although the same type of workflow is running in both countries, e.g., the contract with

external supplier workflow, the organizational policies for the approval step differ. The U.S. version of the organizational policy specifies the vice president of manufacturing is the only eligible user, and the German version specifies that the *Geschäftsführer* is the only eligible user.

Domains were introduced to deal with the issue of autonomous policies. A domain is an abstract entity of management. Organizational policies as well as workflows are related to domains. The previous example might involve two domains: "USA" and "GERMANY." (The domains could be further subdivided.)

The principles just discussed guided the PRA design. As mentioned in the previous section, PRA defines the concepts of organization schema, organizational policy, and a formal language to model policies. In addition, PRA defines interfaces for an execution engine and their use by a WFMS. A detailed discussion of the PRA components follows.

Organization Schema and Organization Structure

The PRA organization schema is a set of objects and relationships that can be freely defined, thus enabling users to model arbitrary organizations. Each member of the set can be instantiated to populate an organization schema, that is, to produce an organization structure. PRA allows users to define constraints on the organization structure to avoid erroneous structures. For example, if an enterprise has the policy that an employee must not report to more than two people, PRA enables the user to define a constraint that specifies that one person can be related to only two others through a "reports to" relationship. If a modeler adds a third reporting line, the system detects the violated constraint.

Organizational Policy

An organizational policy specifies a set of eligible users for a given workflow, which can be either elementary (a step) or composite. A set of users is not stable and therefore fixed but specified through an expression called an organizational expression. An organizational expression specifies the selection of users with particular properties from an organization structure. For example, an expression might enumerate users, select all users able to play a particular role, or select a user related to some other in a specific way. Additionally, organizational expressions can refer to the history of a workflow or to its internal data, such as local variables, and thus be dependent on the workflow state. Consequently, the set of users for the same step in two different instances of the same workflow might be different. Consider, for example, the travel expense reimbursement workflow, with the user selection for the sign step dependent on the authorization level. In two instances of the workflow, the amounts to be reimbursed might differ such that different people, e.g., the manager and the vice president, must execute the two sign steps.

To provide a general mechanism for determining a set of eligible users for a workflow, PRA organizational policies accommodate operations in addition to executing a step or taking responsibility for a composite workflow. Delegating a workflow and undoing a workflow are two examples. To delegate a workflow, an organizational policy has to ensure that both the person who delegates the workflow and the person to whom the workflow is assigned are eligible users. The operation of undoing a workflow (i.e., to undo the results achieved thus far) and starting again can result in wasted effort and unrecoverable work. Therefore, a WFMS must carefully choose eligible users for this operation.

To deal with various workflow operations, a PRA organizational policy relates a workflow type and one of its operations in a given domain to an organizational expression. An organizational policy is defined as the tuple <workflow type, operation, domain, organizational expression>. For example, the organizational policy for the fill step in the travel expense reimbursement example is <TravelExpenseReimbursement.Fill, execute, USA, 'every user who plays the role of employee'>. Since an applicant should be able to undo the step and start again, the WFMS must also specify the organizational policy <TravelExpenseReimbursement.Fill, undo, USA, 'the user who started fill'>. (The next section describes PRA's formal language for specifying organizational policies.)

When a WFMS determines that a workflow in a particular domain is to be executed, it calls the policy resolution engine, which looks for the appropriate organizational policy and evaluates its organizational expression. The engine returns the results of the evaluation, i.e., the set of eligible users, to the WFMS, which subsequently assigns the workflow to those users. One organizational policy can be reused for several workflow types, domains, etc., by entering a set in the appropriate element of the tuple. For example, if the organizational

policy for the fill step of the travel expense reimbursement workflow is the same in the U.S. as it is in Europe, the policy could be modeled as <TravelExpenseReimbursement.Fill, execute, {USA, EUROPE}, 'every user who plays the role of employee'>.

Policy Definition Language

From the organizational viewpoint, the following elements are necessary to run a workflow: an organization schema together with its instantiation, the organizational policies for this workflow, and the relevant organizational expressions. To describe these elements in a formal way, PRA defines a language called the Policy Definition Language (PDL), which consists of several parts. The first part enables the definition of an organization schema and its population. The second part is concerned with organizational expressions. Finally, the third part supports the definition of organizational policies.

The following figures illustrate the PDL for a sample organization schema and organization structure, some organizational expressions, and some organizational policies for the travel expense reimbursement workflow. Figure 5 shows the PDL for the organization schema displayed in Figure 2a. The PDL for the instantiation displayed in Figure 2b appears in Figure 6.

The organization schema definition part of the PDL looks like a data definition language (DDL) in a relational database. Two differences exist, though: (1) PDL distinguishes organizational object types from organizational relationship types, and (2) PDL allows complex data types (e.g., sets as attributes). If a policy resolution engine is built on top of a relational database, a compiler or a translator within

```
ORGANIZATION_TYPE Role
        ATTRIBUTES name: String
                   authorization_level: set(task, amount);
        KEYS name;

ORGANIZATION_TYPE Group
        ATTRIBUTES name: String
        KEYS name;

ORGANIZATION_TYPE User
        ATTRIBUTES name: String
                   office_tel_#: String
                   e_mail: String
                   absence: {vacation, ill, business, available}
        KEYS name;

RELATIONSHIP_TYPE Reports_to
        FROM User
        TO   User
        ATTRIBUTES kind: {line, functional, none}

RELATIONSHIP_TYPE Plays
        FROM User
        TO   Role
        ATTRIBUTES duration_from: date
                   duration_to: date

RELATIONSHIP_TYPE Responsible_for
        FROM User
        TO   Group

RELATIONSHIP_TYPE Belongs_to
        FROM User
        TO   Group

Note that, for simplicity, we assume user names to be unique. In reality,
this is not the case and the modeling must deal with nonunique names.
```

Figure 5 Policy Definition Language for the Sample Organization Schema Shown in Figure 2a

```
Role "Employee",  {}
     "Manager",   {(TravelExpenseReimbursement.Sign, 1000),
                   (CapitalEquipmentOrder.Sign, 5000)}
     "FinancialClerk", {}
     "Secretary", {}
     "Engineer",  {}
     "VP",        {(TravelExpenseReimbursement.Sign, *),
                   (CapitalEquipmentOrder.Sign, *)}

Group "Sales"
      "Manufacturing"
      "Engineering"
      "Administration"

User "Al",      "[1] 125-5589", "al@center.com",      available
     "Nina",    "[1] 125-5590", "nina@center.com",    available
     "Ken",     "[1] 125-5601", "ken@center.com",     available
     "Susan",   "[1] 125-5609", "susan@center.com",   business
     "Matt",    "[1] 125-4499", "matt@center.com",    available
     "Charles","[1] 125-4580", "charles@center.com", available
     "Mike",    "[1] 125-0101", "mike@center.com",    available

Reports_to "Al",      "Nina", line
           "Ken",     "Nina", line
           "Nina",    "Mike", line
           "Susan",   "Matt", line
           "Charles", "Matt", line
           "Matt",    "Mike", line
           "Mike",    "",     none

Plays "Al",      "Employee",  01-02-88, 0-0-0 (* open ended *)
      "Al",      "FinancialClerk", 01-02-88, 0-0-0
      "Nina",    "Employee",  01-02-90, 0-0-0
      "Nina",    "Manager",   01-02-90, 0-0-0
      "Ken",     "Employee",  01-02-91, 0-0-0
      "Ken",     "Secretary", 01-02-91, 0-0-0
      "Susan",   "Employee",  01-02-92, 0-0-0
      "Susan",   "Secretary", 01-02-92, 0-0-0
      "Matt",    "Employee",  01-02-88, 0-0-0
      "Matt",    "Manager",   01-02-88, 0-0-0
      "Charles", "Employee",  01-02-88, 0-0-0
      "Charles", "Engineer",  01-02-88, 0-0-0
      "Mike",    "Employee",  01-02-90, 0-0-0
      "Mike",    "VP",        01-02-93, 12-31-97

Responsible_for "Al",   "Sales"
                "Al",   "Manufacturing"
                "Al",   "Engineering"
                "Mike", "Sales"
                "Mike", "Manufacturing"
                "Mike", "Engineering"

Belongs_to "Al",      "Administration"
           "Nina",    "Engineering"
           "Ken",     "Administration"
           "Susan",   "Administration"
           "Matt",    "Engineering"
           "Charles", "Engineering"
           "Mike",    ""
```

Figure 6 Policy Definition Language for the Sample Organization Structure (Instantiation) Shown in Figure 2b

the engine translates the organization schema definition part of PDL into a set of DDL statements.

Figure 7 lists the organizational expressions required to formulate the organizational policies for the travel expense reimbursement workflow. Note that the organizational expression for employees selects all users who play the role of employee. The RETURNS statement indicates the search for users. The definition of the plays relationship type in Figure 5 indicates that the employee is of the type role. This information is sufficient to formulate a query to the underlying database system in an implementation of a policy resolution engine.

The PDL for the organizational policies for the travel expense reimbursement example appears in Figure 8. The WFMS applies the first organizational policy when assigning the fill step in a travel expense reimbursement workflow. The policy is valid in three domains, USA, EUROPE, and ASIA, for the execute operation, which has no parameters. The policy engine returns a set of all users who are able to play the role of employee. The second policy listed in Figure 8 returns a set of all users who play the role of secretary and who report to the same user as the applicant.

Independent from the travel expense reimbursement example are the sample separation of duty and delegation policies shown in Figures 9 and 10. The organizational policy that specifies separation of duty ensures that the user who signs the expense form is different from the user who fills out the form. The policy that models the delegation operation contains a parameter that specifies to which person the sign step is to be delegated. Only the manager of the applicant can call this operation and then only if the parameter specifies either the next higher manager or the responsible vice president. The step can be delegated only to one of these two users.

Since the PDL is well defined, it can be used not only by designers to model organizations and policies but also by developers of graphics-oriented tools. Such tools could present graphical symbols to users to be manipulated. When a user decides to commit the changes, the tool generates a PDL script, which is fed into the policy resolution engine.

```
ORGANIZATIONAL_EXPRESSION employees()
    RETURNS User: user
        user plays employee

ORGANIZATIONAL_EXPRESSION secretaries()
    RETURNS User: user
        user plays secretary

ORGANIZATIONAL_EXPRESSION manager_of(User: a_user)
    RETURNS User: user
        a_user reports_to user

ORGANIZATIONAL_EXPRESSION subordinates_of(User: a_user)
    RETURNS User: user
        user reports_to a_user

ORGANIZATIONAL_EXPRESSION group_of(User: a_user)
    RETURNS Group: group
        a_user belongs_to group

ORGANIZATIONAL_EXPRESSION VP_responsible_for_group_of(User: a_user)
    RETURNS User: user
        user plays VP
        INTERSECTION
        user responsible_for group_of(a_user)

ORGANIZATIONAL_EXPRESSION executing_agent(Workflow: a_workflow)
    RETURNS User
        (* provided by the historical services of WFMS *)
```

Figure 7 Organizational Expressions for the Travel Expense Reimbursement Example

```
ORGANIZATIONAL_POLICY
        WORKFLOW  TravelExpenseReimbursement.Fill
        OPERATION Execute()
        DOMAIN    USA, EUROPE, ASIA
        ORGANIZATIONAL_EXPRESSION employees()

ORGANIZATIONAL_POLICY
        WORKFLOW  TravelExpenseReimbursement.Check
        OPERATION Execute()
        DOMAIN    USA, EUROPE, ASIA
        ORGANIZATIONAL_EXPRESSION
                  secretaries()
                    INTERSECTION
                  subordinates_of(
                    manager_of(
                      executing_agent(
                        TravelExpenseReimbursement.Fill)))

ORGANIZATIONAL_POLICY
        WORKFLOW  TravelExpenseReimbursement.Sign
        OPERATION Execute()
        DOMAIN    USA, EUROPE, ASIA
        ORGANIZATIONAL_EXPRESSION
                  manager_of(
                    executing_agent(
                      TravelExpenseReimbursement.Fill))

ORGANIZATIONAL_POLICY
        WORKFLOW  TravelExpenseReimbursement.Reimburse
        OPERATION Execute()
        DOMAIN    USA, EUROPE, ASIA
        ORGANIZATIONAL_EXPRESSION
                  financial_clerks()
                    INTERSECTION
                  User: user responsible_for
                  group_of(
                    executing_agent(
                      TravelExpenseReimbursement.Fill))
```

Figure 8 Organizational Policies for the Travel Expense Reimbursement Example

```
ORGANIZATIONAL_POLICY
        WORKFLOW  TravelExpenseReimbursement.Sign
        OPERATION Execute()
        DOMAIN    USA, EUROPE, ASIA
        ORGANIZATIONAL_EXPRESSION
                  manager_of(
                    executing_agent(
                      TravelExpenseReimbursement.Fill))
                  DIFFERENCE
                  executing_agent(
                      TravelExpenseReimbursement.Fill)
```

Figure 9 Organizational Policy for the Separation of Duty

Approaches like the ones mentioned earlier in the paper provide a fixed set of types for modeling an organization or a fixed set of functions, such as "role player" or "supervisor," from which to select users for a workflow. None of these approaches provides a language like PDL that can freely define the organizational aspect as the application semantics requires.

```
ORGANIZATIONAL_POLICY
        WORKFLOW  TravelExpenseReimbursement.Sign
        OPERATION Delegate (User: a_user)
        DOMAIN    USA, EUROPE, ASIA
        ORGANIZATIONAL_EXPRESSION
                  IF a_user IN
                     (manager_of(
                        manager_of(
                          executing_agent(
                            TravelExpenseReimbursement.Fill)))
                      OR
                      VP_responsible_for_group_of(
                        executing_agent(
                          TravelExpenseReimbursement.Fill)))
                  THEN
                     manager_of(
                       executing_agent(
                         TravelExpenseReimbursement.Fill))
```

Figure 10 Organizational Policy for the Delegate Operation

Policy Resolution Engine

The policy resolution engine is a mechanism that evaluates organizational policies for a WFMS. Serving as a base service, the policy resolution engine manages organizational policies and organizational expressions, as well as the organization schema and its population. The engine also provides interfaces for the definition, modification, and evaluation of these objects. The interfaces are distinguished by the kind of service they provide. There are basically two kinds of interfaces: evaluation interfaces and management interfaces.

Evaluation Interfaces Policy resolution engine clients use evaluation interfaces to evaluate organizational policies or organizational expressions when necessary. The engine provides four evaluation interfaces: two for organizational policies ("resolve" and "conform to") and two for organizational expressions (also "resolve" and "conform to"). The resolve operation for organizational policies expects a workflow reference and one of its operations as input values. This operation selects an appropriate organizational policy, evaluates it, and returns a set of users eligible to execute the given task of the workflow. The conform to operation for organizational policies expects a workflow reference, one of its operations, and a user as input values. This operation resolves the appropriate organizational policy for the workflow and checks whether the user is contained in the set of results for that organizational policy (i.e., if the user conforms to the policy). If the user is contained in the set of results, the conform to operation returns the value "true"; otherwise it returns the value "false." Policy resolution engine clients use this operation to validate a request by a user to execute a certain task of a workflow.

The resolve and conform to operations for organizational expressions work analogously. Instead of a workflow reference, the operations expect the name of an organizational expression as input. The operations evaluate the named organizational expression and return the set of results, which is used if the resolve operation is called. The conform to operation returns true and false values as described in the previous paragraph.

Management Interfaces Management interfaces are used to define, modify, or delete organizational policies, organizational expressions, or organization schemas and their populations. These interfaces look like the following operations that are provided for organizational policies: create, delete, modify, list, get. The create operation creates an organizational policy; the delete operation deletes a policy; the modify operation allows users to change an organizational policy to adjust to new requirements; the list operation returns the identifiers of all policies; and the get operation returns the complete description of a policy.

Designers do not call these management interfaces directly, since they communicate their changes through user-friendly interfaces or tools. These tools are either graphics oriented or language oriented. In a graphics-oriented tool, a designer

manipulates icons and graphical symbols, which in turn results in calls to the appropriate management interfaces. Alternatively, a graphics tool can generate a PDL script according to the manipulations of a user and submit this script to the policy resolution engine. In this case, the engine interprets the submitted script and changes its internal state accordingly. Language-oriented tools enable a designer to directly express changes using PDL. These tools take specifications and translate them into management interface calls. Of course, they can also submit the language specifications directly as PDL scripts to the policy resolution engine, as described above.

Legacy Databases Many large enterprises have developed databases that contain some or all of the organizational data the policy resolution engine needs to evaluate organizational policies. These databases, called legacy databases, might be self-implemented or based on standards efforts like those related to providing directory services on networks, i.e., X.500.[74] In general, organizations must deal with one of the following scenarios:

- No legacy database exists. No existing database has to be considered, and the policy resolution engine can use its own database to build up organizational knowledge.
- Legacy databases contain all relevant data. To use the policy resolution engine, the database must provide a sufficiently expressive query interface, on top of which queries issued from the engine can be evaluated. The only additional information that has to be stored is organizational policies and organizational expressions. The organization has to choose whether to extend the legacy databases or to use the database within the policy resolution engine.
- A legacy database contains some relevant data. In addition to organizational policies and organizational expressions, organizational objects and relationships must be stored in either the legacy database or the database of the policy resolution engine.

If the relevant data is stored in several databases, the querying interface must be built in such a way that the policy resolution engine can issue the necessary queries, which might span several databases. Furthermore, semantics issues have to be dealt with in heterogeneous environments.[75,76]

Architectural Considerations—Clients of a Policy Resolution Engine From an architectural point of view, there are two possible ways to design a policy resolution engine:

1. Incorporate the policy resolution engine into a WFMS. The engine would be a module, whose operations are hidden by the exported interfaces of the WFMS. All calls to the engine operations would be made through the interface of the WFMS.
2. Make the policy resolution engine an independent component. The engine would be a server with a WFMS system as one of its clients. All clients of the engine, including the WFMS, would be able to directly access the exported operations of the engine.

PRA recommends the implementation of a policy resolution engine as an independent base service, which can be used by clients other than a WFMS. For example, an electronic mail system can be a client of the policy resolution engine. Since electronic mail is sent to users, rather than enumerate the electronic mail addresses of the recipients by hand, organizational expressions can provide the addresses. For example, a manager could send an electronic mail message to "all my subordinates" or an engineer could send an electronic mail message to "all my colleagues who are engineers." The sample operational expression shown in Figure 11 returns all electronic mail addresses of all subordinates of a given user.

Another possible client is a transaction processing monitor, which incorporates workflow management.[77] Dayal et al. reference a service called role resolution, which is an earlier development of policy resolution.[78]

```
ORGANIZATIONAL_EXPRESSION subordinates(User: a_user)
        RETURNS String: user.e_mail
                user reports_to a_user
```

Figure 11 Organizational Expression for Electronic Mail

Figure 12 shows a schematic representation of a policy resolution engine with three clients—a WFMS, a transaction processing monitor, and an electronic mail system.

Summary

The sample workflow discussed in this paper, that is, the travel expense reimbursement workflow, illustrates that roles are sufficient as task assignment rules for only the simplest scenarios. Since workflow management systems are deployed in situations where complex workflows are modeled and executed, a more general and powerful model called the Policy Resolution Architecture (PRA) was developed. PRA provides the concept of an organizational policy. An organizational policy is more general than a role in that it relates a workflow type to an organizational expression that determines the set of eligible users for the workflow. Because they state all criteria a user has to fulfill and do not limit the selection based on their properties or interrelationships, organizational policies specify all eligible users. Since an organizational expression is related to a workflow type by an organizational policy, task assignment through organizational policies is a very general approach. Organizational policies are evaluated based on organization schema and their populations (organization structures). Since PRA provides a way to model arbitrary complex organization schemas, arbitrary organizations can be modeled and subsequently populated. This generality, in conjunction with organizational policies, provides a powerful and flexible approach to task assignment in workflow management.

Acknowledgments

I want to thank the anonymous referees whose remarks helped me a great deal in revising this paper. Susan Thomas assisted me by improving my English and thus making the paper more readable. Kathy Stetson was always very helpful in coordinating the writing and revision processes.

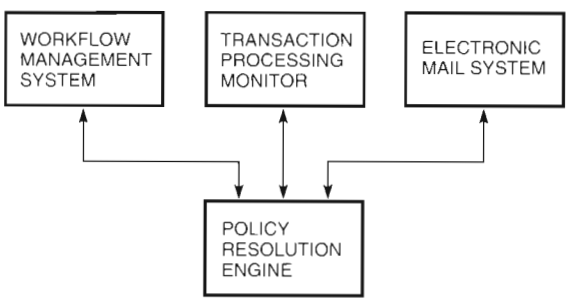


Figure 12 Client-server Structure of a Policy Resolution Engine

References

1. T. May, "Know Your Work-Flow Tools," *BYTE* (July 1994).
2. T. Kreifelts and P. Seuffert, "Addressing in an Office Procedure System," *Message Handling Systems, State of the Art and Future Directions: Proceedings IFIP WG 6.5 Working Conference on Message Handling Systems,* R. Speth, ed. (Amsterdam: North-Holland, 1987).
3. S. Chang and W. Chan, "Transformation and Verification of Office Procedures," *IEEE Transactions on Software Engineering,* vol. SE-11, no. 8 (August 1985).
4. W. Croft and L. Lefkowitz, "Task Support in an Office System," *ACM Transactions on Office Information Systems,* vol. 2, no. 3 (July 1984).
5. C. Ellis and G. Nutt, "Office Information Systems and Computer Science," *Computing Surveys,* vol. 12, no. 1 (March 1980).
6. C. Ellis and M. Bernal, "Officetalk-D: An Experimental Office Information System," *First SIGOA Conference on Office Information Systems* (1982).
7. C. Ellis, "Formal and Informal Models of Office Activity," *Information Processing 83,* R. Mason, ed. (Amsterdam: North-Holland, 1983).
8. B. Karbe and N. Ramsperger, "Concepts and Implementation of Migrating Office Processes," *Verteilte Künstliche Intelligenz und Kooperatives Arbeiten: 4. Internationaler GI-Kongreß Wissensbasierte Systeme, Informatik Fachberichte 291,* W. Brauer and D. Hernandez, eds. (Munich: Springer-Verlag, October 1991).
9. T. Kreifelts, "Coordination of Distributed Work: From Office Procedures to Customizable Activities," *Verteilte Künstliche Intelligenz und Kooperatives Arbeiten: 4. Internationaler GI-Kongreß Wissensbasierte Systeme, Informatik Fachberichte 291,* W. Brauer and D. Hernandez, eds. (Munich: Springer-Verlag, October 1991).

10. C. Cook, "Streamlining Office Procedures—An Analysis Using the Information Control Net Model," *AFIPS Conference Proceedings of the 1980 National Computer Conference,* Anaheim, California (May 1980).

11. I. Ladd and D. Tsichritis, "An Office Form Flow Model," *AFIPS Conference Proceedings of the 1980 National Computer Conference,* Anaheim, California (May 1980).

12. L. Baumann and R. Coop, "Automated Workflow Control: A Key to Office Productivity," *AFIPS Conference Proceedings of the 1980 National Computer Conference,* Anaheim, California (May 1980).

13. M. Zisman, "Representation, Specification and Automation of Office Procedures," Ph.D. dissertation (Philadelphia: University of Pennsylvania, Wharton School, 1977).

14. B. Curtis, M. Kellner, and J. Over, "Process Modeling," *Communications of the ACM,* vol. 35, no. 9 (September 1992).

15. W. Deiters and V. Gruhn, "The Funsoft Net Approach to Software Process Management," *International Journal of Software Engineering and Knowledge Engineering,* vol. 4, no. 2 (1994).

16. W. Deiters, V. Gruhn, and H. Weber, "Software Process Evolution in MELMAC," *The Impact of CASE Technology on Software Processes Series on Software Engineering and Knowledge Engineering,* vol. 3, D. Cooke, ed. (Singapore: World Scientific Publishing, 1994).

17. D. Harel et al., "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *Proceedings of the Tenth International Conference on Software Engineering* (1988).

18. W. Humphrey and M. Kellner, "Software Process Modeling: Principles of Entity Process Models," *Proceedings of the Eleventh International Conference on Software Engineering* (May 1989).

19. M. Jaccheri and R. Conradi, "Techniques for Process Model Evolution in EPOS," *IEEE Transactions on Software Engineering* (December 1993).

20. T. Katayama, "A Hierarchical and Functional Software Process Description and Its Enaction," *Proceedings of the Eleventh International Conference on Software Engineering* (May 1989).

21. P. Mi and W. Scacchi, *Operational Semantics of Process Enactment and Its Prototype Implementations* (Los Angeles: University of Southern California, Computer Science Department, April 1991).

22. P. Mi and W. Scacchi, *Modeling Articulation Work in Software Engineering Processes* (Los Angeles: University of Southern California, Computer Science Department, April 1991).

23. P. Mi and W. Scacchi, "A Knowledge-Based Environment for Modeling and Simulating Software Engineering Processes," *IEEE Transactions on Knowledge and Data Engineering,* vol. 2, no. 3 (September 1990).

24. L. Osterweil, "Software Processes Are Software Too," *Proceedings of the Ninth International Conference on Software Engineering* (March/April 1987).

25. L. Williams, "Software Process Modeling: A Behavioral Approach," *Proceedings of the Tenth International Conference on Software Engineering* (1988).

26. W. Royce, "Managing the Development of Large Software Systems," *Proceedings of the Ninth International Conference on Software Engineering* (March/April 1987).

27. B. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM Software Engineering Notes,* vol. 11, no. 4 (August 1986).

28. C. Hegarty and L. Rowe, "Control Loops and Dynamic Run Modifications Using the Berkeley Process-Flow Language," *Proceedings of the Third International Conference on Data and Knowledge Systems for Manufacturing and Engineering,* Lyons, France (1992).

29. S. Jablonski, "Data Flow Management in Distributed CIM Systems," *Proceedings of the Third International Conference on Data and Knowledge Systems for Manufacturing and Engineering,* Lyons, France (1992).

30. *Proceedings of the Third International Conference on Data and Knowledge Systems for Manufacturing and Engineering,* Lyons, France (1992).

31. H. Yoshikawa and J. Goossenaerts, eds., *Information Infrastructure Systems for Manufacturing* (Amsterdam: North-Holland, 1993).

32. T. Härder and A. Reuter, "Principles of Transaction-oriented Database Recovery," *ACM Computing Surveys,* vol. 15, no. 4 (December 1983).

33. P. Attie, M. Singh, A. Shet, and M. Rusinkiewicz, "Specifying and Enforcing Intertask Dependencies," *Proceedings of the Nineteenth International Conference on Very Large Databases (VLDB),* Dublin, Ireland (1993).

34. Y. Breitbart, A. Deacon, H. Schek, and G. Weikum, "Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows," *SIGMOD Record,* vol. 22, no. 3 (September 1993).

35. U. Dayal, M. Hsu, and R. Ladin, "A Transactional Model for Long-Running Activities," *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB),* Barcelona, Spain (September 1991).

36. H. Garcia-Molina and K. Salem, "Sagas," *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (1987).

37. *Bulletin of the Technical Committee on Data Engineering,* vol. 16, no. 2 (June 1993).

38. S. Jablonski, "Transaction Support for Activity Management," *Proceedings of the Workshop on High Performance Transaction Processing Systems (HPTS),* Asilomar, California (1993).

39. H. Wächter and A. Reuter, "The ConTract Model," in *Transaction Models for Advanced Database Applications*, A. Elmagarmid, ed. (San Mateo, California: Morgan Kaufmann, 1992).

40. T. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," *ACM Computing Surveys,* vol. 26, no. 1 (March 1994).

41. T. Malone, K. Crowston, J. Lee, and B. Pentland, "Tools for Inventing Organizations: Toward a Handbook of Organizational Processes," CCS WP #141, Sloan School WP #3562-93 (Cambridge: Massachusetts Institute of Technology, Center for Coordination Science, May 1993).

42. R. Burkhart, "Process-based Definition of Enterprise Models," *Proceedings of the First International Conference on Enterprise Integration Modeling Technology (ICEIMT),* Hilton Head, South Carolina (June 1992).

43. C. Bußler, "Enterprise Process Modeling and Enactment in GERAM," *Proceedings of the International Conference on Automation, Robotics and Computer Vision (ICARCV '94),* Singapore (November 1994).

44. M. Fox, "The TOVE Project: Towards a Common-Sense Model of the Enterprise," *Proceedings of the First International Conference on Enterprise Integration Modeling Technology (ICEIMT),* Hilton Head, South Carolina (June 1992).

45. *Proceedings of the First International Conference on Enterprise Integration Modeling Technology (ICEIMT),* Hilton Head, South Carolina (June 1992).

46. R. Katz, "Business/enterprise Modeling," *IBM Systems Journal,* vol. 29, no. 4 (1990).

47. J. Sowa and J. Zachman, "Extending and Formalizing the Framework for Information Systems Architecture," *IBM Systems Journal,* vol. 31, no. 3 (1992).

48. F. Vernadat, "Business Process and Enterprise Activity Modelling: CIMOSA Contribution to a General Enterprise Reference Architecture and Methodology (GERAM)," *Proceedings of the International Conference on Automation, Robotics and Computer Vision (ICARCV '94),* Singapore (November 1994).

49. T. Williams, "Architectures for Integrating Manufacturing Activities and Enterprises," *Information Infrastructure Systems for Manufacturing,* H. Yoshikawa and J. Goossenaerts, eds. (Amsterdam: North-Holland, 1993).

50. F. Flores and T. Winograd, *Understanding Computers and Cognition* (Reading, MA: Addison-Wesley, 1987).

51. R. Medina-Mores, R. Winograd, T. Flores, and F. Flores, "The Action Workflow Approach to Workflow Management Technology," *Proceedings of the ACM 1992 Conference on Computer Supported Cooperative Work (CSCW)*, Toronto, Ontario, Canada (November 1992).

52. T. Danielsen and U. Pankoke-Babatz, "The Amigo Activity Model," in *Research into Networks and Distributed Applications*, R. Speth, ed. (Munich: North-Holland, Elsevier Science Publishers B.V., 1988).

53. R. Fehling, K. Joerger, and D. Sagalowicz, *Knowledge Systems for Process Management* (Palo Alto, CA: Teknowledge Inc., 1986).

54. J. Guyot, "A Process Model for Data Bases," *SIGMOD Record*, vol. 17, no. 4 (December 1988).

55. C. Bußler and S. Jablonski, "An Approach to Integrate Workflow Modeling and Organization Modeling in an Enterprise," *Proceedings of the Third IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE)*, Morgantown, West Virginia (April 1994).

56. S. Jablonski, "MOBILE: A Modular Workflow Model and Architecture," *Proceedings of the Fourth Working Conference on Dynamic Modelling and Information Systems*, Noordwijkerhout, Netherlands (September 1994).

57. M. Hsu, A. Ghoneimy, and C. Kleissner, "An Execution Model for an Activity Management System," *Proceedings of the Workshop on High Performance Transaction Systems* (1991).

58. M. Hsu and M. Howard, "Work-Flow and Legacy Systems," *BYTE* (July 1994).

59. F. Leymann and W. Altenhuber, "Managing Business Processes as an Information Resource," *IBM Systems Journal*, vol. 33, no. 2 (1994).

60. *Workflow Management Software: The Business Opportunity* (Ovum Reports, December 1991).

61. T. White and L. Fischer, "*New Tools for New Times: The Workflow Paradigm* (Alameda: Future Strategies Inc., Book Division, 1994).

62. J. Bair, "Contrasting Workflow Models: Getting to the Roots of Three Vendors," *Proceedings of the Groupware '93 Conference*, San Jose, California (1993).

63. S. Sarin, K. Abbot, and D. McCarthy, "A Process Model and System for Supporting Collaborative Work," *Proceedings of the ACM SIGOIS Conference on Organizational Computing Systems* (November 1991).

64. M. Shan, "Pegasus Architecture and Design Principles," *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, D.C. (May 1993).

65. M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth, "Using Flexible Transactions to Support Multi-System Telecommunication Applications," *Proceedings of the Eighteenth International Conference on Very Large Databases (VLDB)*, Vancouver, British Columbia, Canada (1992).

66. D. Evans, "Putting Elves to Work: Workflow Technology in a Law Firm," *Proceedings of the Groupware '93 Conference*, San Jose, California (1993).

67. D. Sng, "A National Information Infrastructure for the 21st Century Collaborative Enterprise," *Proceedings of the International Conference on Automation, Robotics and Computer Vision (ICARCV '94)*, Singapore (November 1994).

68. R. Marshak, "Characteristics of a Workflow System—Mind Your P's and R's," *Proceedings of the Groupware '93 Conference*, San Jose, California (1993).

69. C. Bußler and S. Jablonski, "Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems," *Proceedings of the Fourth International Workshop on Research Issues in Data Engineering: Active Database Systems (RIDE-ADS '94)*, Houston, Texas (February 1994).

70. C. Ellis, S. Gibbs, and G. Rein, "Groupware—Some Issues and Experiences," *Communications of the ACM*, vol. 34, no. 1 (January 1991).

71. L. Lawrence, "The Role of Roles," *Computers and Security*, vol. 12, no. 1 (1993).

72. L. Aiello, D. Nardi, and M. Panti, "Modeling the Office Structure: A First Step towards the Office Expert System," *Second ACM SIGOA Conference on Office Information Systems (ACM SIGOA)*, vol. 5, nos. 1 and 2 (1984).

73. D. Denning, *Cryptography and Data Security* (Reading, MA: Addison-Wesley, 1983).

74. *Blue Book, Volume VIII, Fascicle VIII.8, Data Communication Networks: Directory, Recommendations X.500-X.521 (Study Group VII)*, Comité Consultatif Internationale de Télégraphique et Téléphonique.

75. S. Ceri and J. Widom, "Managing Semantic Heterogeneity with Production Rules and Persistent Queues," *Proceedings of the Nineteenth Conference on Very Large Databases (VLDB)*, Dublin, Ireland (1993).

76. W. Kent, "Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language," *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, Barcelona, Spain (September 1991).

77. U. Dayal et al., "Third Generation TP Monitors: A Database Challenge," *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, D.C. (May 1993).

78. C. Bußler, "Capability Based Modeling," *Proceedings of the First International Conference on Enterprise Integration Modeling Technology (ICEIMT)*, Hilton Head, South Carolina (June 1992).

Stewart V. Hoover
Gary L. Kratkiewicz

The Design of DECmodel for Windows

The DECmodel for Windows software tool represents a significant advance in the development of business process models. The DECmodel tool allows rapid development of models and graphical representations of business processes by providing a laboratory environment for testing processes before propagating them into workflows. Such an approach can significantly reduce the risk associated with large investments in information technology. The DECmodel design incorporates knowledge-based, simulation, and graphical user interface technology on a PC platform based on the Microsoft Windows operating system. Unique to the design is the manner in which it separates the model of the business processes from the views or presentations of the model.

Many approaches have been developed for understanding, specifying, testing, and validating business processes. In the late 1980s, Digital began to reengineer some of its most complex and mission-critical business processes. It soon became apparent that modeling methodologies and tools were needed to document, test, and validate the reengineered processes before they were implemented, as well as to provide a high-level specification for their design and implementation. Consequently, Digital decided to provide the business process engineer with tools similar to those used by architects, mechanical designers, and computer and software engineers.

The first implementation of Digital's dynamic business modeling technology, Symbolic Modeling, was developed at Digital's Artificial Intelligence Technology Center. The technology was embodied in an application called Symmod, which in 1991 ran only on a VAXstation system.[1] Symmod's knowledge base and simulation engine were implemented using the LISP programming language and the Knowledge Craft product, a frame-based knowledge representation package with modeling and simulation features.[2] Because models were written in LISP code, users had to be computer programmers as well as business consultants. The application contained a graphical presentation builder and viewer implemented in the C programming language that used a relational database for presentation storage. The user had to start the knowledge base component and the presentation component as separate processes. A primitive mailbox system was used for interprocess communication. To serve the needs of nontechnical business users and to achieve the necessary product quality, Symmod needed to be completely redesigned and rebuilt.

In early 1991, the Modeling and Visualization Group decided to build a product version of the Symmod application, which would be released as the DECmodel tool. The team drafted requirements, specifications, and an architecture. The DECmodel product was initially targeted at two platforms: VAXstation workstations running under the DECwindows operating system and personal computers (PCs) running under the Windows NT operating system. As users were interviewed and requirements were accumulated, it became clear, however, that by far the most important platform for DECmodel users was the PC platform based on the Windows operating system. Consequently, the DECmodel development effort shifted to this platform.

During 1991, the team enhanced the existing version of Symmod so that it would meet user needs until the release of the product version for PCs. The most significant enhancement was the development of an X Window System interface for building and editing models. A second important enhancement was a graphical shell program that transparently started up the knowledge base and presentation components for the user.

In March 1992, Digital officially announced Phase 0 (the strategy and requirements determination phase) of the DECmodel for Windows product.

Design and Development Goals

The DECmodel product design team had the following goals:

- Provide a modeling tool that maps directly to business processes
- Allow the modeling of both the static and the dynamic characteristics of the business process
- Allow multiple views of the business process model by separating the model from the presentation of the business process during simulation
- Allow the user to interact with the tool and to make decisions while the business process is being simulated in order to let the user "test-drive" the business process
- Provide a tool that is easy to use for business consultants and that requires no programming

Note that the designers intentionally omitted the following goals from the DECmodel design:

- Include resource constraints and queuing
- Allow the user to perform a statistical analysis of the behavior of the business process

By far the most important goal for the DECmodel design was the first one listed, an obvious mapping between elements of the model and business processes. The anticipated users of the DECmodel tool were business analysts and consultants, not system designers and software engineers. The designers felt that adding levels of abstractions to a modeling tool would make it less acceptable to the intended users. A notable corollary to providing an obvious mapping was modeling both the static and the dynamic characteristics of the business process.

To engage the user in interacting with the model and test-driving the business process required a graphical interface that was separate from the model. This "presentation" layer of the DECmodel tool provides a layout and graphical appearance that has the look and feel of the actual business process, hiding the irrelevant technical details of the model. The presentation enables the user to step through the business, watching information and material flows occur, and thus see where the dependencies and concurrencies exist.

Designers believed that while simulating the business, the user should be able to interact with the model and thereby select and test more than one scenario. The DECmodel tool was intended to be a working scale model of the business, giving the user a sense of how the business process would work as different choices were made. The tool, by design, neither predicts congestion and throughput as a function of resource constraints nor provides information through statistical reports. The DECmodel product was designed to provide a slow, deliberate simulation of the business, not to compress weeks or years of activities into a few seconds, leaving behind only a statistical summary.

The team's development goals for the DECmodel product were to

- Provide a tool that runs on a popular hardware platform used by business consultants
- Achieve a short time-to-market, i.e., delivery within one year
- Utilize a widely accepted software base technology (for maintainability)

The DECmodel World View

Every modeling and simulation tool is based on a predefined view of the world.[3] In the DECmodel world view, a business process is composed of aggregate centers capable of performing one or more tasks or work steps. Each aggregation is referred to as a process, and the tasks that can occur in a process are called activities. Processes communicate through the exchange of messages, which are sent by activities and received by another process or other processes or by the same process that contains the activity.[4]

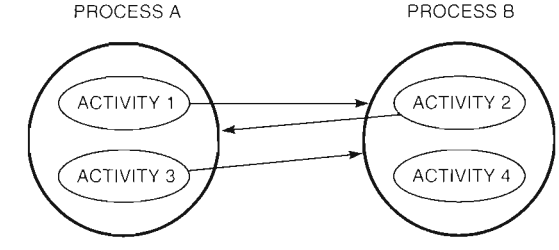
This view differs significantly from the one taken by the typical workflow model in which work steps are directly linked. In the DECmodel model, an activity that sends a message to a process has no knowledge of what work steps will occur next. For example, when a customer (a process) sends an order (a message) to a supplier (another process), the customer does not know what work steps (activities) the supplier will initiate when it receives the order. It is invisible to the customer whether or not the supplier decides to change its work rules, for instance, by sending the order to a second source because materials are not available. Similarly, when the supplier's activities have been completed and the material that was ordered has

been sent to the customer, the supplier has neither knowledge of nor dependencies on the work steps that the customer undertakes next. In contrast, in a workflow model each task is directly linked to another task. Changes in the supplier's way of doing business force changes in how the customer's tasks connect to the supplier's tasks. More succinctly, the DECmodel tool encapsulates the behaviors and work rules of each individual process in the larger business process. This difference between the process and workflow models is shown in Figure 1.

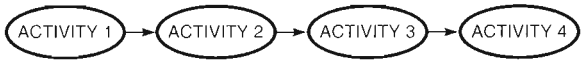
Processes, Activities, and Messages

As described above, the DECmodel model represents a business process as a collection of smaller encapsulated processes. The behavior of each process is defined by the activities that it contains. The DECmodel tool provides three general types of activities: generating activities, processing activities, and terminating activities. Generating and terminating activities represent the boundaries of the model; processing activities represent the work steps in the business process.

An activity is characterized by (1) a receive rule, which defines the messages that the activity needs for initiation, (2) a duration, and (3) a send rule, which defines the messages that the activity sends out at the end of its duration. Generating activities have only send rules, and terminating activities have only receive rules.



(a) Process Model



(b) Workflow Model

Figure 1 The Process Model versus the Workflow Model

Activities can send messages to processes only. The receiving process makes the message known to every activity that uses the message in its receive rule. Messages are universal to the model, and the same message type can be sent by activities in different processes.

Processes can have state knowledge (attributes) that can be assigned values as a side effect of an activity being completed. The activity can use a process attribute value to decide what messages to send out and where to send them. That is, processes have a state that can be altered to change the behavior of the model.

Like processes, messages can contain information, which is stored in their attributes. When a process receives a message and passes it on to an activity, information in the message can be used in both the receive rule and the send rule of the activity. Additionally, the information in a received message can be copied into the attributes of any message that an activity sends. In this way, the DECmodel tool supports information propagation.

The DECmodel representation of business borrows heavily from both the stochastic-timed Petri net (STPN) model and the object paradigm found in object-oriented design.[5,6]

The Stochastic-timed Petri Net Model versus the DECmodel Model An STPN model represents a system as a collection of places, transitions, arcs, and tokens. Places contain tokens and act as inputs to transitions. A transition results in the movement of a token to another place if an arc exists between the transition and the place. Before a transition can occur, a token must be present at each place that is connected to the transition by an arc. Associated with each transition is an exponentially distributed random variable that expresses the delay between the enabling of the transition and the firing of the transition.

The DECmodel model welds the STPN place, transition, and arc elements into a single object called an activity. The analogous elements of the STPN and DECmodel models are

STPN	DECmodel
Place	Activity receive rule
Transition	Activity duration
Token	Message
Arc	Activity send rule
—	Process

The DECmodel model goes beyond the STPN model by

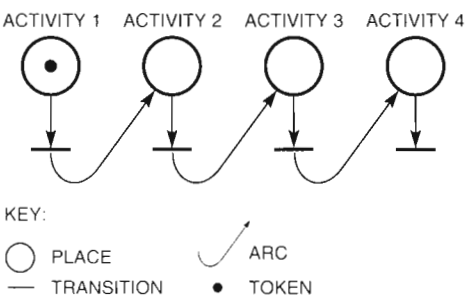
1. Adding the process object between the activity send rules (arcs) and the activity receive rules (places). Each process can have multiple activity send rules. As the process object receives messages (tokens), it dispatches them to the appropriate activity receive rule (place).
2. Allowing more than one type of message (token) to exist.
3. Storing information in both the processes and the messages (tokens).
4. Using AND, OR, and message-matching receive rules in the activity receive rules (places).
5. Not restricting durations to being exponentially distributed random variables.

Like an STPN model, a DECmodel model does not explicitly have resources but can represent the availability of a resource by sending a message to a process when the resource is available.

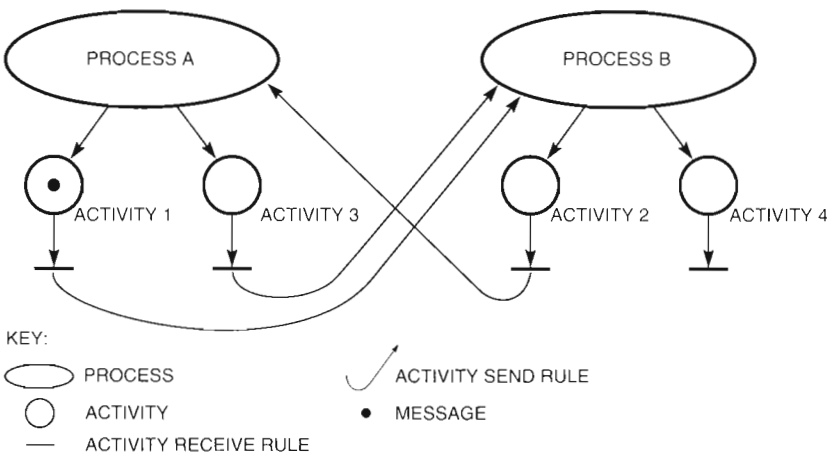
Figure 2 shows the workflow system from Figure 1 as both an STPN model and a DECmodel model with the process receiving messages from the activities.

The DECmodel Model and Object-oriented Design

The elements of object-oriented design that the DECmodel model fully draws upon are encapsulation of information and the message-method paradigm. Information is encapsulated within DECmodel



(a) Stochastic-timed Petri Net Model of a Four-activity Workflow



(b) A DECmodel Model of a Four-activity Workflow with a Process Dispatching Messages between Activities

Figure 2 The Stochastic-timed Petri Net Model versus the DECmodel Process-activity Model

objects and is not available globally. However, an important difference exists between DECmodel systems and object-oriented systems. In DECmodel systems, a number of messages may by required to trigger a behavior; whereas, in classical object-oriented systems, each message triggers a method.

The DECmodel tool supports polymorphism, in that the same message can be sent to different processes, which can result in different behaviors. Developers investigated going beyond standard polymorphism by using one message to trigger different activities within the same process. The approach considered was to use process "filters" to examine the information in a message and then decide which activity or activities in the process should receive it. This feature was not completely developed because of time constraints and a less-than-clear mapping between the concept and the actual practices in most business. Further, using activity send rules that utilize the information contained in messages can provide a similar capability.

The DECmodel tool does not support inheritance, but the underlying technology of the product does support this feature. As in the case of nonstandard polymorphism, time-to-market pressures and the lack of clear evidence that the feature would be used in business processes drove the decision not to include inheritance support. Also, the DECmodel product does not currently support class types beyond the built-in classes of the process and the three activity types.

Process Hierarchies

To address the goal of having a strong mapping between the model and real business processes, the DECmodel model supports processes within processes. Processes can receive messages in two ways: hierarchical routing and peer-to-peer routing.

In a business process, a message sent to a high-level process should travel through the process hierarchy to the activity that is to act upon the message. For example, an activity in the sales process should be able to send a message to the manufacturing process and not be concerned that manufacturing contains several subprocesses. The knowledge of how to relay a message should be in the receiving process, not the sending process.

In business, however, much communication occurs on a peer-to-peer basis, with information seldom routed up and down the organization hierarchy. For example, the results of a marketing research activity go directly to the manufacturing planning function without traveling down through the various levels of the manufacturing organization. In a DECmodel model, as in most businesses, when an activity is completed, a message can be sent directly to any process in the business.

The DECmodel design feature that allows processes to receive messages and then pass them on to subprocesses and activities can result in multiple message receipts for a single send operation. That is, one activity can send a single message that is received by every activity in the model that includes the message in its receive rule. Modeling experts disagree about how well this phenomenon maps to real business processes. The DECmodel user can avoid this effect, if desired, by using uniquely named messages in the send rules of activities.

The Presentation

The first DECmodel design goal was supported by the modeling paradigm of processes, activities, and messages. The presentation aspect of the DECmodel tool supports the goals of a strong separation between the model and the graphical representation of the business process and the need to support user interaction and decisions during model simulation.

The presentation of the model is based on views that contain networked nodes. Each node in a view can represent zero or more processes in the model; however, no process can be represented by more than one node in a single view. This mapping between the processes in the model and the nodes in a view allows the user to develop and animate multiple views of the model simultaneously. For example, one view may show the model at its lowest level of detail, with each process in the model mapped to a single node. Another view may show a higher level of mapping, with multiple processes mapped to the same node. A third view may map processes based on attributes such as geographic location, the organizational chart, or technology. The construction of the views is left to the creativity of the analyst building the model.

During model simulation, the DECmodel tool uses animation to show the movement of messages from one process to another. The user can also view the messages and their attributes.

To accommodate user interaction, the DECmodel tool provides a menu send rule in the definition of an activity. If an activity uses the menu send rule, just before the activity fires, a menu appears that allows the user to make a choice that determines what messages are to be sent by the activity and

which processes are to receive them. The user is unaware of the actual send rule; the choice made forces one of a set of send rules to be selected. The use of menus, animation of messages moving between processes, and user-controlled stepping through the simulation gives the user the feeling of test-driving the business process.

Architecture and Development Process

The overall DECmodel architecture, shown in Figure 3, contains two layers. The inner layer of the architecture is the internal engine, which provides the means for representing, storing, and executing (simulating) models. The internal engine is designed using ROCK, a frame-based, object-oriented knowledge representation system, and AMP, a modeling and simulation frame-class library implemented in ROCK.[7] The outer layer of the architecture is the user interface, which provides the means for all user interaction with the DECmodel model and has two major components: the model builder and the presentation builder. The user interface is designed as a set of classes specialized from the Microsoft Foundation Classes. Interaction between the two layers is achieved with an internal application programming interface (API).

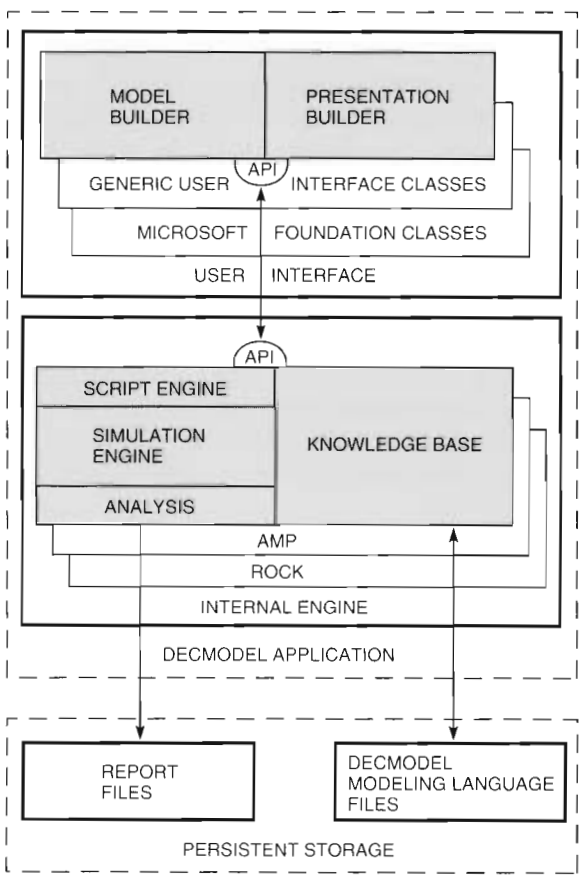


Figure 3 DECmodel Architecture

This architecture was chosen for both technical and pragmatic organizational reasons. The partitioning into two layers allowed the internal engine to be built using state-of-the-art knowledge representation technology and the user interface to be built using state-of-the-art graphical user interface technology. The disadvantages in this separation were the necessity of designing an internal API and the need to duplicate some data (nominally stored in the knowledge base) in the user interface.

The partitioning mapped well to the human resources available in the DECmodel team. The DECmodel engineers had strong skills in developing LISP, knowledge-based, and X Window System applications but little experience in developing C++, ROCK, or Microsoft Windows applications. With the architectural separation, one team was able to focus on the internal engine using C++ and ROCK and, therefore, did not have to learn much about Windows programming. The other team was able to focus on the user interface using C++ and Windows programming tools and did not have to learn anything about ROCK. The engineering team felt that the efficient use of human resources in the development process overcame the technical disadvantages of the approach.

DECmodel development proceeded with the two teams. Since the bulk of their development work was completed first, the members of the knowledge base team also worked on the user interface team toward the end of the development process.

Design and Implementation

This section describes the design of the two DECmodel layers: the internal engine and the user interface.

Internal Engine

The internal engine represents models of dynamic business processes in a knowledge base and executes these models using discrete event simulation. This layer provides a set of services for interacting with the knowledge base. These services are accessed through the DECmodel tool's internal API. The internal engine contains the DECmodel knowledge base, simulation engine, and means of persistent storage. Using the DECmodel methodology to

represent and execute business process models, the internal engine

- Represents the structure, attributes, and behavior descriptions of the business processes in a knowledge base. (This representation is the model.)
- Represents the structure, attributes, and behavior descriptions of the animated visualization of the model in a knowledge base. (This representation is the presentation.)
- Represents the connections between the model and the presentation in a knowledge base. (This representation is the model-presentation mapping.)
- Represents the dynamic behavior of the business processes by allowing for discrete event simulation of the knowledge base.

Knowledge Base The DECmodel knowledge base contains the frame-based, object-oriented representation of the model, the presentation, and the connections between them. It also maintains the model relations, attributes, and methods. The knowledge base contains both classes and instances. The classes specify DECmodel objects; sets of instances make up specific models and presentations. In addition to containing all the information about model and presentation behavior and structure, the knowledge base contains all the graphical information used by the model builder and the presentation builder. This information is updated in real time.

Knowledge Representation Technology The DECmodel knowledge base and simulation engine are implemented in ROCK, a frame-based, object-oriented knowledge representation system written in the C++ programming language. ROCK implements the IMKA knowledge representation technology and is used as a set of API functions in a C++ programming environment.

ROCK provides useful features such as frames, multiple inheritance of data and methods, user-defined relationships, and contexts. The basic unit of knowledge in ROCK is a frame, which represents an object or a concept. A frame is a collection of slots that contain the attribute, relationship, and procedural information about the object or the concept. Attribute slots store values, relation slots store user-defined links between frames, and message slots store methods (functions) that are executed when the frame receives the appropriate message from the application program. Class frames represent object types or categories. Instance frames represent particular members of a class. ROCK requires frame classes to be organized in a class hierarchy. Attribute slots and message slots can inherit values and methods from classes at a higher level in the hierarchy. This mechanism can be used to define default values for frame classes. Both frame classes and frame instances are objects, and both can be dynamically created, operated on, and deleted during run time. With respect to the C++ language, all frames appear to have the same data type. Slots are objects, whose behavior is defined independent of the frames.

Portions of the knowledge base are built using AMP, a modeling and simulation frame-class library implemented in ROCK. AMP contains objects for representing process models that contain entity flow, for constructing and running discrete-event simulations, and for generating, collecting, and reducing statistical data.

The DECmodel frame classes are subclasses of ROCK and AMP classes and contain relations, attributes, and methods that describe the content and behavior of DECmodel objects. Some DECmodel frame classes are abstract classes used only as a basis for more specific subclasses; others are used for instantiation of DECmodel objects. The DECmodel tool contains three types of frame classes: model objects, presentation objects, and project objects. A specific DECmodel project is represented within the knowledge base as a set of model, presentation, and project instances. These instances are created in the knowledge base by loading a DECmodel modeling language (DML) file or through interaction with the model builder or the presentation builder.

Persistent Storage The DML is a subset of the ROCK frame definition language and is used by the knowledge base for persistent storage. A DECmodel project is stored as ASCII text in three files that contain the model, presentation, and mapping objects. The language employs ROCK syntax but uses only the frame classes and slots defined in the DECmodel knowledge base.

The DECmodel tool utilizes the ROCK frame definition interpreter as the DML interpreter. Since the ROCK interpreter was not intended to be used for persistent storage, the DECmodel developers made several minor modifications to obtain the desired

error handling capabilities. The DECmodel tool contains its own DML code generator.

Simulation Engine The simulation engine executes a discrete event simulation of the model in the knowledge base. This simulation can be performed either interactively or in a batch mode. The simulation engine was designed to be so robust that a model can be simulated at any stage of its development, regardless of inconsistencies or undefined elements.

The simulation engine interacts with the presentation builder to control simulation, animation, and graphics. The user controls simulation through the presentation builder. The presentation builder calls simulation engine API functions to perform the requested actions, such as starting, stepping through, pausing, ending, and reinitializing the simulation.

Script Engine and Compiler Scripts provide a means of specifying user-defined actions to customize model animation and to perform special presentation actions during simulation. The DECmodel tool contains a language for defining scripts, a script compiler, and a script engine for executing the scripts. Although the DECmodel team wanted to avoid requiring any programming in the tool, developers decided that a script language was the only way to implement these features in the available time frame.

The script language contains functions for

- Annotating, hiding, showing, flashing, moving, highlighting, and scaling presentation icons
- Playing sounds and sound loops
- Animating connections between nodes
- Showing, hiding, and clearing certain kinds of windows
- Starting other applications
- Temporarily stopping execution
- Loading a new project
- Starting and pausing the simulation
- Displaying files
- Displaying a list of DECmodel development team members

Analysis and Reporting Services The knowledge base contains services that allow the user to analyze models and presentations in the knowledge base and to generate reports.

The consistency advisor checks models, presentations, and mappings for inconsistencies and potential problems at any point in the model development process. This check is analogous to the syntax check performed by a compiler. The consistency advisor check is the primary model-building debugging aid for users. Inconsistencies in the model do not prevent a model from being simulated.

The model description report lists the description, messages sent, and messages received for each activity and process. The model table report contains the basic model information in a table format for easy access by another application, database, or spreadsheet. The simulation summary report contains information on simulation performance.

Design and Implementation Decisions The internal engine for the first DECmodel product release, DECmodel for Windows version 1.0, was implemented as a Windows dynamic link library (DLL) using the Windows version of ROCK version 1.0, the Windows version of AMP version 1.0, and Microsoft C/C++ version 7.0. For DECmodel for Windows version 1.1, developers ported the internal engine to Microsoft Visual C++ version 1.0.

Several options existed for implementing the DECmodel knowledge base. The knowledge base of the Symmod application, the precursor to the DECmodel product, was implemented in a LISP environment. The DECmodel engineering team wanted to move to a more standard programming environment and, therefore, focused on C++ and C++-based tools. However, a straight C++ implementation would have required the reimplementation of knowledge representation, simulation, and modeling technology available in other tools.

Another modeling and simulation technology, the Modeling and Simulation System (MSS), had been developed for Digital's Artificial Intelligence Technology Center by the Carnegie Group, Inc. (CGI).[8] This graphical tool was designed at a lower level than Symmod. It used a modeling simulation language and was developed to implement the next version of Symmod. However, the MSS modeling paradigm was not compatible with that of the DECmodel tool.

IMKA had also been recently developed by CGI, funded by a consortium of companies, as a

replacement for the Knowledge Craft product. IMKA's implementation, ROCK, lacked some of the class libraries included in Knowledge Craft for simulation and process modeling but ran significantly faster than Knowledge Craft. The engineering team decided to use ROCK to implement the knowledge base because of its knowledge representation power and its C++ compatibility. Digital contracted with CGI to port the class libraries to ROCK. The team, therefore, had a head start in designing and implementing the internal engine. The portability of ROCK was also an advantage; switching to the Windows platform from the DECwindows platform caused no disruption in development.

The original intent of the engineering team was to implement the DECmodel tool as a single executable file. The knowledge base contains much global data, however, and restrictions on the number of data segments required developers to implement the internal engine as a DLL. This encapsulation of the internal engine allows it to be used in other applications and enables easy porting to other platforms. The DECmodel team developed a set of internal API functions and structures to allow interactions between the DLL-based internal engine and the executable-based user interface.

The Symmod application had a modeling language based on LISP for persistent storage of models and used a relational database for persistent storage of presentations. Consideration was given to developing a modeling language specific to the DECmodel tool. Instead, the engineering team decided to use the ROCK frame definition language, since it was already completely defined and debugged and had an interpreter. The team used this language for persistent storage of both models and presentations to allow easy sharing of projects between users and to simplify debugging by users and DECmodel developers.

The knowledge base team was responsible for implementing the internal API between the user interface and the knowledge base. This interface was specified in detail early in the project. The team kept the specification up-to-date throughout the project. It prepared 19 revisions and produced a final document of more than 200 pages. This specification kept interface problems to a minimum, thus defusing a potential source of major technical problems.

The team specified the objects in great detail early in the project. It also held several internal and external design reviews. These measures reduced the number of potential design problems and thus yielded a higher-quality product and a faster implementation.

User Interface

The user interface provides the means for all user interaction with the DECmodel tool. It has two major components: the model builder and the presentation builder.

The user interface is designed as a set of classes specialized from the Microsoft Foundation Classes. Most of these special DECmodel user interface classes correspond to frame classes in the knowledge base; the remainder are necessary for implementing the user interface. The three main types of user interface classes—windows, graphic objects, and dialog boxes—are used by both the model builder and the presentation builder.

Window Classes The user interface contains several types of window classes: graphics windows, text windows, and a frame window.

The graphics window classes are all derived from the generic DECmodel graphics window class. Graphics windows contain graphic objects, such as boxes or lines. Users act upon these windows through menu commands or through the Windows messages generated by the mouse and mouse buttons. The graphics windows are the model window, the view windows, and the palettes. Menu commands specific to each graphics window are handled by message handlers within the window class.

The text window classes are derived from the generic DECmodel text window class. Text windows are generally read-only and display various types of textual information, such as descriptions, the text of files, and clock information. As in the case of graphics windows, menu commands specific to each text window are handled by message handlers within the window class.

The one frame window class, i.e., the top window class, is derived from the CMDIFrameWnd Microsoft Foundation Class and serves as the frame window for the application. The menu commands not specific to a particular window are handled by default message handlers within this window.

Graphics Classes Graphics window classes use graphic objects to build models and presentations. These classes implement the processes, activities, nodes, connections, and annotations displayed in the Model Editing Window and in the views.

Dialog Box Classes The DECmodel tool contains a large number of dialog boxes derived from the CModalDialog Microsoft Foundation Class. The tool uses these dialog boxes to define the information and relationships contained in the DECmodel objects.

Menus The DECmodel tool uses a set of menus individualized to match the capabilities of the window currently in use. When a user starts the DECmodel application, the tool presents a reduced menu that allows the user to start a new project or to load an existing one. Once a project is in memory, the menu changes as the user switches between the Model Editing Window, the views, and the other windows. Menu commands activate message handler functions within the window classes.

Appearance of the User Interface Figure 4 shows a small but typical DECmodel model. The figure displays each process and its member activities. Note that each of the three activity types is denoted by a different icon. Lines indicate the potential flow of messages. Figure 5 shows the DECmodel presentation for the model that appears in Figure 4. The presentation contains both a view and the supporting windows, e.g., the simulation clock and the description windows.

Design and Implementation Decisions The team implemented the user interface for DECmodel for Windows version 1.0 using Microsoft C/C++ version 7.0 and Microsoft Foundation Classes version 1.0. For DECmodel for Windows version 1.1, developers ported the user interface to Microsoft Visual

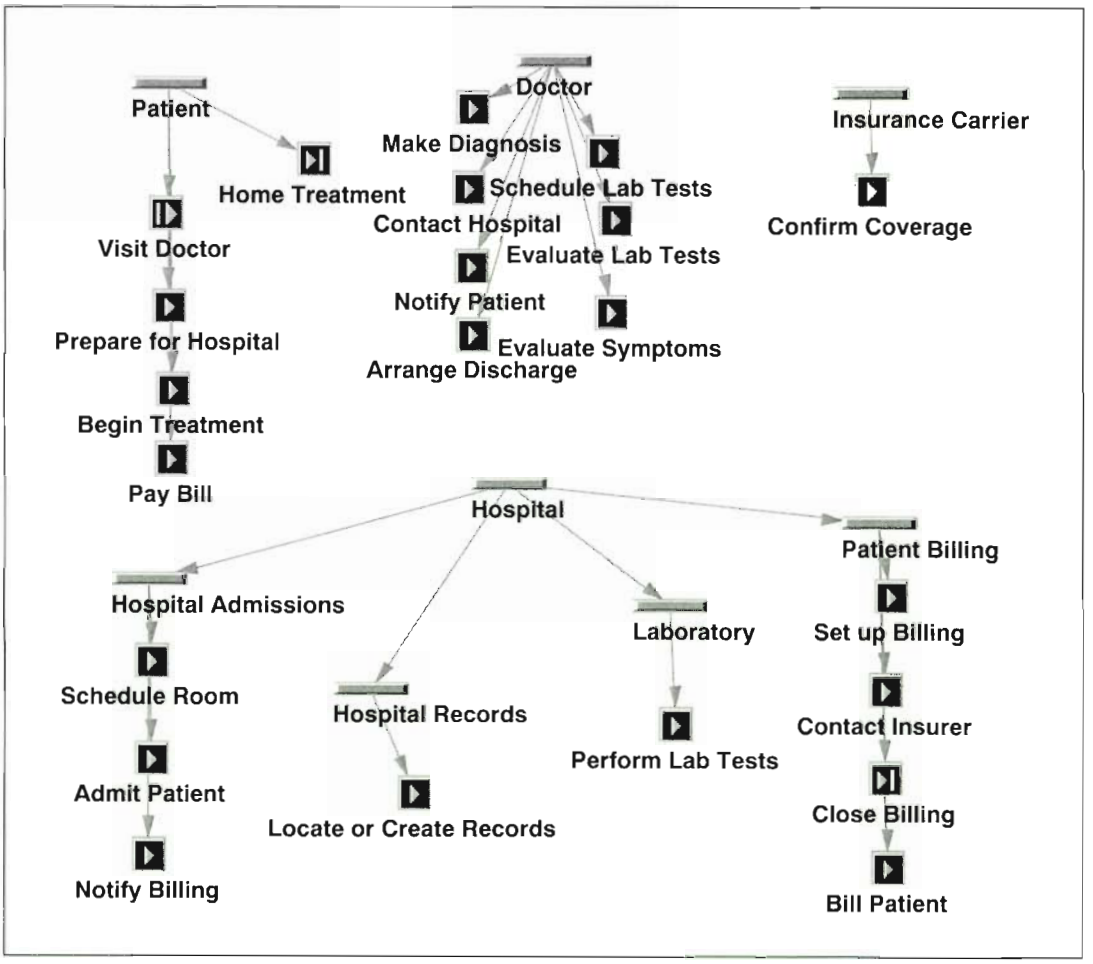


Figure 4 Typical DECmodel Model

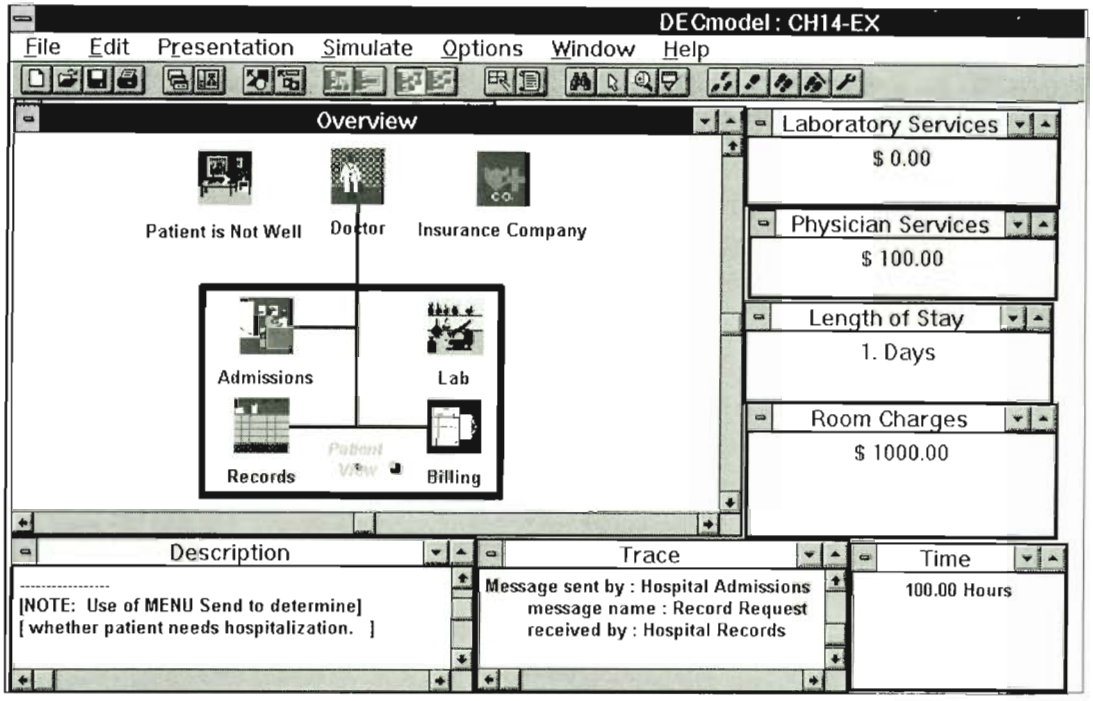


Figure 5 Typical DECmodel Presentation (for the Model Shown in Figure 4)

C++ version 1.0 and Microsoft Foundation Classes version 1.5.

As stated at the beginning of the paper, the DECmodel product was initially targeted at both VAXstation workstations running under the DECwindows operating system and PCs running under the Windows NT operating system. Consequently, when developers decided to focus solely on the PC platform running under the standard Windows operating system, the user interface development effort was disrupted. Engineers had done a significant amount of design work toward achieving a DECwindows implementation.

The DECmodel engineering team considered other class libraries and user interface implementation packages (such as XVT), but most were deficient in Windows features or in the look and feel. Since the Windows operating system was the only platform for the foreseeable future, the engineering team felt that using Microsoft Foundation Classes was the best choice. However, they made this decision after they had performed a significant amount of development work with one of the tools. Much of the work had to be redone, which contributed to the schedule delay.

During the design and development of the DECmodel product, the team debated how graphical to make the user interface, that is, to what extent dialog boxes should be used. Although the goal was to make the user interface as graphical as possible, the tight schedule forced the team to postpone plans for graphical editors in favor of dialog boxes, which were faster to implement. For example, the team had initially planned to implement an Activity Editing Window and had partially developed it. This window was to provide a complete view of an activity and allow graphical editing of its information. Schedule constraints required the team to abandon this plan and to develop a set of dialog boxes that were not as easy to use but were faster to implement.

The user interface design was not specified or committed to storyboards in any detail at the beginning of the project, partially to save time after the disruptions in the development work. This decision led to more lost time later in the project, though, because user interface features were designed quickly and sometimes incompatibly, and consequently required reworking. In addition, the resulting user interface was not as easy to use as it could have been if better planned.

External review of the user interface design was not performed until late in the project. The review yielded some ideas that would have resulted in a more usable product; however, there was not enough time left in the schedule to implement them.

Delivery

A discussion of the released product and the team's success in achieving the design and development goals follows.

Release

Digital released version 1.0 of the DECmodel for Windows product in November 1993 and version 1.1 in April 1994. Version 1.0 contained the basic capabilities for building models and presentations of business processes; version 1.1 added a set of minor enhancements and bug fixes. Because of its small, focused market and the large cost savings that can result from its use, the DECmodel tool was introduced as a low-volume, high-priced product. The product includes the software, example models, documentation, and a week of hands-on training. The DECmodel tool is an integral part of Digital Consulting's reengineering practice.

Success of Design Choices

The separation of the model from the presentation is the single most important element of the product's success. This feature, along with animation, distinguishes the DECmodel tool from its competition. Some users have even requested the capability of building the presentation first and then generating the corresponding model. Such capability would require considerable investigation.

The paradigm of process-activity encapsulation is difficult for some users to become accustomed to. Many still prefer to build a model using a workflow approach, which the DECmodel tool can support, rather than by defining each process and its behavior independently.

The exclusion of resource constraints has limited the application of the DECmodel tool to system design, thus preventing its use in modeling system performance. Although the capability was originally not a product goal, many users would like a future version of the DECmodel product to provide this feature.

To perform special user-defined actions during the simulation, a script language was included in the DECmodel tool. This design feature violated the goal of requiring no programming, and some users found scripts hard to use. However, many users have requested that a future DECmodel version provide more script functions and extend the script language to be more like the BASIC programming language.

Also, to enhance the use of the DECmodel tool in the design of business processes, a future version should support classes to make generic processes available as building blocks of a business process.

Development Successes and Lessons

The DECmodel engineering team successfully released a software product on the Microsoft Windows platform, the one most popular with business consultants. This achievement was significant because the group of engineers began the project with no PC experience. The team did not meet its one-year delivery goal, and the goal slipped to one and one-half years after the Phase 0 announcement. However, this time frame was still extremely short for developing a complex PC product from scratch.

The product retained the existing Symbolic Modeling paradigm (i.e., a process-activity-message model and a strong distinction between model and presentation) and exhibited performance an order of magnitude better than that of the Symmod product, which it replaced. The product utilized the most widely accepted modern programming technology base (C/C++), which simplified maintainability and reduced the need for special training of maintainers.

Splitting the development team into two subteams worked well. It distributed the amount of learning about new technologies required by the engineers and minimized the overall development time. Key factors in the success of this approach were the detailed object and internal API specifications that were kept up-to-date throughout development and thus provided a reliable interface between the two parts of the project.

After the product was released, the DECmodel team identified certain factors that could have made the team and the product even more successful. The entire engineering team would have benefited from Windows training at the onset of the project. The Windows design of the user interface should have been specified and committed to storyboard in much greater detail much earlier in the project. In addition, the team should have arranged for Windows experts to review the design. These changes in the engineering process would have helped the team produce a cleaner, easier-to-use,

more maintainable user interface and would have reduced implementation time. The project schedule should have been created using a bottom-up rather than a top-down process. The initial one-year schedule was based on an unrealistic, management-imposed release date. When the engineering team revised the schedule and calculated a release date based on their detailed estimates, the team met the new date.

Summary

Modeling and simulating business processes is an important part of business process reengineering. Digital developed the DECmodel tool specifically for this type of simulation. Although it borrows many ideas from other disciplines of modeling and simulation, as well as from object-oriented design, the DECmodel product is unique in the way it models business processes, separates the model from the presentation, and represents the model as frames in a knowledge base.

Acknowledgments

The authors would like to acknowledge the following people who also contributed to the design of the DECmodel product: Ty Chaney, David Choi, Laurel Drummond, Peter Floss, Amal Kassatly, Mike Kiskiel, Kip Landingham, and Janet Rothstein.

References

1. *Symmod User's Guide* (Maynard, MA: Digital Equipment Corporation, 1990).
2. *Knowledge Craft Reference Manual* (Pittsburgh, PA: Carnegie Group, 1988).
3. S. Hoover and R. Perry, *Simulation, A Problem Solving Approach* (Reading, MA: Addison-Wesley, 1989).
4. *DECmodel for Windows: Modeler's Guide* (Maynard, MA: Digital Equipment Corporation, 1994).
5. J. Peterson, *Petri Net Theory and Modeling of Systems* (Englewood Cliffs, NJ: Prentice-Hall, 1981).
6. G. Booch, *Object Oriented Design* (Redwood City, CA: Benjamin-Cummings, 1991).
7. *ROCK Software Functional Specification, Version 2.0* (Pittsburgh, PA: Carnegie Group, 1991).
8. *Modeling and Simulation System User's Guide* (Pittsburgh, PA: Carnegie Group, 1991).

Dennis G. Giokas
John C. Rokicki

The Design of ManageWORKS: A User Interface Framework

The ManageWORKS Workgroup Administrator for Windows software product is Digital's integration platform for system and network management of heterogeneous local area networks. The ManageWORKS product enables multiple, heterogeneous network operating system and network interconnect device management from a single PC running under the Microsoft Windows operating system. The ManageWORKS software is a user interface framework; that is, the services it provides are primarily targeted at the integration of the user interface elements of management applications. It manifests the organizational, navigational, and functional elements of system and network management in a coherent whole. Viewers, such as the hierarchical outline viewer and the topological relationships viewer that are components of the ManageWORKS software, provide the organizational and navigational elements of the system. Management applications developed by Digital and by third parties through the ManageWORKS Software Developer's Kit provide the functional elements to manage network entities. This paper discusses the user interface design that implements these three elements and the software system design that supports the user interface framework.

The ManageWORKS Workgroup Administrator for Windows software product is Digital's strategic tool for providing system and network management of heterogeneous local area networks (LANs). It serves as Digital's platform for the integration of PC LAN management. From the perspective of the end user, i.e., the LAN system administrator and network manager, the ManageWORKS product comprises a suite of modules that integrates a diverse set of management activities into one workspace. From the perspective of the developer of system and network management applications, the ManageWORKS product is an extensible and flexible software framework for the rapid development of integrated management modules, all of which presents a consistent user interface.

The design of the management system was user centric, i.e., usability was the top priority. Thus, we began the design work without any preconceived notions about the management software system design. The design that emerged and that is documented in this paper was driven solely by the user interface paradigm developed and tested with our customers.

This paper focuses on how the ManageWORKS software presents and integrates its functionality to the end user. Specifically, the paper presents details of the user interface paradigm and discusses the design rationale and the design methods employed. The paper also discusses the design of ManageWORKS software in support of the user interface framework.

Driving Forces behind the Design

The ManageWORKS software was first released as a component of the PATHWORKS version 5.0 for DOS and Windows product. The foci for that PATHWORKS release set the tone for the ManageWORKS design. The PATHWORKS version 5.0 design objectives were to

1. Enhance the usability of the PATHWORKS product. Since the PATHWORKS system was rooted in a command line–based user interface, the goal was to develop a graphical user interface for the system that was based on the Microsoft Windows operating system. Such a user interface would be contemporary, easier to learn, and easier to use.

2. Enhance the manageability of the PATHWORKS product. The goal was to reduce the cost of ownership by improving the installation, configuration, and administration of the system.

The ManageWORKS design team used two voice-of-the-customer techniques to provide more depth and detail for the two high-level product design objectives. First, the team used Contextual Inquiry to determine a customer profile and to develop a clearer statement of the user's work.[1] Then, the team tested user interface prototypes with customers by means of formal usability testing. From 15 to 20 customers and users participated in each of three rounds of usability testing.

Early in the investigation, Contextual Inquiry revealed that the profile of the PATHWORKS system administrator had changed drastically during the five years since the PATHWORKS product was first released. A typical system administrator in the era of PATHWORKS version 1.0 had been a VAX/VMS system manager who inherited the responsibility of installing and managing a PC file and print-sharing product. The interface into the system was a VT-class terminal running command line-based utilities. Today, a system administrator is usually a PC user who is quite familiar with graphical user interfaces. Such an administrator is more likely to be trained in the installation, configuration, and management of PCs and PC networking software than his/her predecessors. This change in the profile encouraged us to shift the PATHWORKS focus from using host-based command line utilities to manage the system to using client-based graphical utilities.

We also profiled the customer network configuration. During the same five years, it changed from a very simple and homogeneous environment with just a few PATHWORKS servers to a medium-to-large heterogeneous PC LAN. At present, configurations comprise network operating systems that consist of Novell NetWare, Microsoft LAN Manager, and Apple AppleShare file and print services, as well as other services that are emerging in the PC LAN environment. The network operating systems are deployed on their native platforms and by Digital on the OpenVMS and DEC OSF/1 platforms. Each system has its own tools to manage the clients and the servers. Each has a different user interface that results in a long learning curve and thus high training costs or low productivity for system administrators. Customers reported that they desired tools with a consistent user interface to manage this diversity.

The team employed software usability testing throughout the development life cycle. Two usability tests were performed with early design prototypes; the final test was performed with our first pass at a detailed concept design. We performed the usability testing with customers to test user interface and functional element design concepts that we developed as a result of the Contextual Inquiry. The user thus served as a design participant. With each iteration of the formal testing, we tested specific functional concepts in three key areas: (1) mechanisms to navigate among the managed entities, (2) mechanisms to organize these entities, and (3) the functional capability inherent in the management directives supported. (Note that, in this paper, the servers, services, and resources managed by means of the ManageWORKS software are collectively referred to as managed entities.) The major lessons that we learned from this testing effort and then applied to the user interface and software designs are as follows:

1. The ManageWORKS software had to provide mechanisms to navigate among a diverse set of managed entities on the LAN or in some user-defined management domain. Users want to be able to view and thus "discover" the entities that are to be managed. The system had to present the managed entities in graphical display formats that were familiar and enticing to users. Users welcome the ability to support different styles of presentation. Finally, users need easy mechanisms to navigate through the hierarchy of an entity.

2. Navigation mechanisms, as just described, work well for novice users but become tedious and constraining for more experienced users, as we could attest to after our experience with the prototypes. The solution that we presented to users allowed them to create custom views of their managed entities, i.e., to organize their management domains. This concept was well received by users during usability testing.

3. The ManageWORKS product had to provide mechanisms that consistently performed the functions that were common among a diverse set of management applications. The product design presents users with an object-oriented view of the managed environment. The building block of this design is the object, an abstraction of a manageable entity such as a server or a network router. Each object is a member of a single

object class that describes the set of object instances within it. The ManageWORKS application renders objects to the user as icons in a viewer. For example, for a LAN that contains three NetWare servers, the object class called NetWare Servers would contain three objects, each of which represents one of the three individual NetWare servers on the LAN. When users focus on an object, the tool reveals which actions are valid in the object's current context. This approach differs from the traditional command line approach in which the user first selects the utility (action) and then specifies the objects upon which to act. Interestingly, whereas novice users found this object-focused concept easy to grasp, those who considered themselves strong users of the traditional command line management utilities experienced difficulty in grasping the new concept.

4. The typical customer has a diverse and large (200 to 1,000) number of entities to manage. To address this need, the prototype testing presented users with the ability to manage more than one entity at the same time and the ability to manage many entities as one. Users liked being able to view and modify the properties of multiple entities at the same time as well as being able to modify the same property across a set of like entities.

5. In addition to providing a consistent user interface, the ManageWORKS product should integrate the management tools into one workspace. User feedback led to the design of the user interface framework as the delivery vehicle for a diverse set of management applications.

The Key Software Design Principles

At this point in the development cycle, the design focus shifted from developing and testing user interface and functionality concepts to designing the ManageWORKS software itself. With what we considered to be a good understanding of the user's needs, we proceeded to design a software architecture to support those requirements.

Prior architectures that were familiar to the design team served as starting points for the design. The following two examples represent sources of design concepts that we employed and adapted to suit our objectives. Each represents an opposing end of the spectrum with respect to design objectives and implementation.

The ManageWORKS team adopted the concept of plug-in modules, a software design that is supported by the Windows Dynamic Link Library (DLL) architecture.[2] The design is also in common use by many Windows applications including the Windows Control Panel, the utility that manages the local desktop's configuration and user preferences.[3]

The next challenge was to decide how much constraint to impose on the design of the ManageWORKS' plug-in modules and how consistent the modules must be. Digital's extensible enterprise management director, the DECmcc product, incorporated some excellent concepts.[4] In particular, our design was influenced by the way in which DECmcc layered the management responsibility into presentation modules, functional modules, and access modules. Early in the design process, we decided to separate the navigation and presentation of managed entities from the access and functional management of the entities.

Another DECmcc concept, which is used, for example, in the access module layer, was the presentation of a consistent view to the layers above.[4] This concept, however, was not suitable for the ManageWORKS design because it would have placed constraints on the user interface design, in particular, on the presentation of the attributes of managed entities. The design team was not willing to compromise on this aspect of the design.

Thus, we decided on a ManageWORKS design that can best be described as a user interface framework. The initial release, which was a component of PATHWORKS version 5.0 for DOS and Windows, offered few services other than to tie together the user interface elements required for system and network management. The user interface services needed were dictated by the five user interface requirements previously described.

The ManageWORKS design incorporates two types of plug-in modules: navigation modules, referred to in the ManageWORKS product as Object Navigation Modules (ONMs), and application modules, referred to as Object Management Modules (OMMs). The ManageWORKS framework controls the control flow and messaging between the modules.

ONMs allow for any number of navigation models to be supported and used singly or simultaneously by the user. Although, by design, ONMs possess no knowledge of the managed entities or entity relationships they display, they do possess the ability to display entities with the relationships inherent in them. ONMs also provide the mechanisms for

browsing and navigating through the management hierarchy. In addition to navigation capabilities, ONMs provide the user interface for organizing entities into a user-defined management domain.

The OMMs are responsible for managing the entities. The OMM design has three key components.

1. OMMs provide the methods used to manage the entities. These methods include the functions of discover, create, view, modify, and delete. The OMMs also have the option of presenting to the user additional methods. That is, since each OMM knows how to manage the entities for which it is responsible, it knows which actions can be applied to an entity based on the entity's current state and the user's context.
2. OMMs provide access to the managed entities. An OMM can use any interprocess communication mechanism to access or to manage an entity. Examples include the task-to-task, remote procedure call, and object request broker mechanisms. Since a PC LAN environment affords no common way for a management director to communicate with all the types of devices present, the design team decided to leave the choice of access mechanism up to the OMM.
3. OMMs provide the user interfaces required for managing the entities. This design component allows developers to present an interface that best suits the needs of the user and best maps to the entity being managed. It also allows for flexibility, evolution, and innovation in the user interface of OMMs. The ManageWORKS design team did not want to impose a user interface style or present a user interface that was compromised by the diversity of applications that we envisioned running within the context of the framework, e.g., by being the least common denominator. Even though one of the key product design goals was a consistent user interface, we felt that it was important to allow the OMMs to control the user interfaces. First, we thought the design benefits outweighed the risk of any inconsistency. Second, we encouraged, but did not enforce, consistency by means of a user interface style guide and common libraries that implemented those guidelines.[5,6]

The plug-in modules also have a residual benefit. Because these modules can easily be added to or removed from the environment, they provide an easy way to extend and to customize the ManageWORKS product. Digital and third parties can develop new ONMs and OMMs and simply enroll them into the system. Users have the additional benefit of being able to customize the product to support only the ONMs and OMMs that are useful in their environment.

The User Interface of ONMs and OMMs

Given the key software design elements presented in the previous section, the focus of the paper now returns to the user interface. This section describes what was implemented to support the customer requirements and the design framework.

The user interface framework manifests the organizational, navigational, and functional elements of system and network management in a coherent whole. For example, the first three menus on the ManageWORKS menu bar—Viewer, Edit Viewer, and Actions—are all the tools the user needs to manage entities. A discussion of the Viewer and Edit Viewer menus follows.

By means of the ManageWORKS Viewer menu, ONMs present display elements, called viewers, to the user. Each instance of a window that an ONM creates is considered a viewer. A ManageWORKS viewer is one of the organizational elements for the user and is the root-level object for navigation. Each viewer is a viewport into a set of managed entities that the user may be browsing and navigating through. A viewer is analogous to a word processor's document, i.e., a viewer is a ManageWORKS "document." Just as you can create new documents and open, close, or edit existing documents when you use a word processing application, you can perform the same functions on viewers when using the ManageWORKS software.

ManageWORKS ONMs are responsible for the navigational and organizational display properties. The current ManageWORKS release comes with two ONMs. One ONM supports a hierarchical display of managed entities. This display is rendered in a single viewer window graphically as a tree or textually as an outline. The other available ONM supports the relational display of managed entities, rendered as a map. The map ONM can also support a hierarchy; each map is rendered in a new viewer instance. Figure 1 shows ManageWORKS with two hierarchical viewer styles and a map viewer. The hierarchical views are the Outline view (shown in the Browser viewer) and the Outline Tree view (shown in the IP Hierarchical View viewer). In addition to the map viewer (shown in the IP Discovery viewer), note the navigation window for the map viewer (shown in

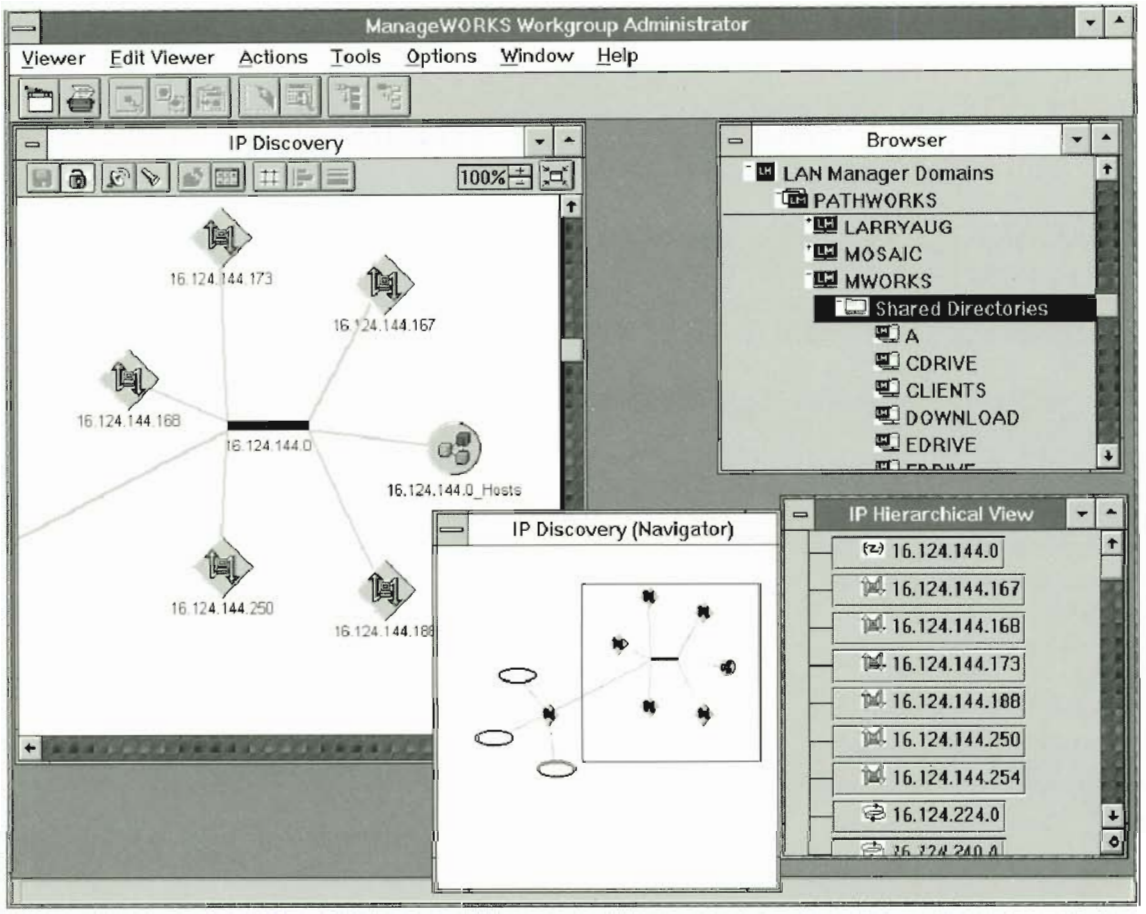


Figure 1 ManageWORKS Viewers

the IP Discovery (Navigator) viewer). This view shows a scaled map; the entire contents of the map viewer appears in a rectangular outline, which represents the user's current viewport into the data. The user can use the PC pointing device to drag and reposition the viewport.

Because the ONM maintains context when the user "edits," i.e., modifies, the contents of a viewer, the user can customize or organize the managed entities as desired. By means of the Edit Viewer, ONMs allow user customization within a viewer with the support of user-definable hierarchies. For example, each instance of a viewer can represent a different management domain for the user. The benefit is that the user can find objects and then arrange them into hierarchies that are most useful.

As stated earlier, OMMs control the user interfaces for displaying and modifying managed entity properties. The ManageWORKS framework provides for consistency in how the OMMs invoke the user interfaces and in how the user interfaces interrelate to the ONMs.

The consistency starts with the ManageWORKS Actions menu. The basic management directives on managed entities originate from this menu. The major challenge in designing this menu was to avoid using too many menu items, menu items that would change constantly (i.e., by addition or deletion), menu items that had three or four levels of hierarchy, and menu items that were not context sensitive to what the user was doing. The objective was to find a small set of words that conveyed the management functions the user would most often perform. We felt that these words should always be present in the Actions menu, but to eliminate confusion for the user, they should be rendered inactive when not valid. On the other hand, we realized that this small set of menu choices could never fully support the actions on managed entities; therefore, the software had to provide an extensibility mechanism.

We began the design process by developing an entity/action matrix. One axis contained a list of the entities that we envisioned being managed from the ManageWORKS software. The other axis contained a list of the actions that could be performed on the entities. We marked the intersections of the axes. In forming the list of actions, we chose words that were used in existing products that managed the same entities, words that we thought should be considered in a good user interface, and finally, synonyms to those words already listed. This approach gave us a clear picture of the common actions and also provided a thesaurus of words from which to choose. The common actions on managed entities that emerged from this exercise were

1. Make a new entity of some type.
2. Display all the managed entities.
3. View and modify the entity's properties.
4. Eliminate the entity.

The ManageWORKS software supports these common actions through the following Action menu choices:

1. Create. Choose Create to make a new entity.
2. Expand. Choose Expand to view all the entities that the ManageWORKS software is managing.
3. Properties. Choose Properties to display a dialog box that manifests all the entity's properties. The user can then view the properties and make modifications, as appropriate.
4. Delete. Choose Delete to eliminate the entity.

The design of the Properties dialog box is one of the key user interface style elements of the ManageWORKS product; however, ManageWORKS does not enforce or provide for this element. Rather, the consistency is a function of a user interface style guide for OMMs and some common library routines that support this user interface style.[5,6] Figure 2 shows the dialog boxes of two of the three OMMs that come with the current ManageWORKS product: the Simple Network Management Protocol (SNMP) Manager OMM and the LAN Manager (LM) server management OMM. (The third OMM, for NetWare servers, is not shown.) Note the Selected Objects field in the SNMP dialog box. The ManageWORKS software allows the user to select multiple objects of the same class from a viewer and to invoke an OMM method. The list of selected objects is contained within this drop-down list box. The user can easily view the attributes of different objects from the same dialog box. The dialog box displays various sets of managed entity properties. The user can select the desired set of properties from the View or Modify drop-down list boxes.

Figure 2 demonstrates that two dialog boxes can be active at the same time. This feature supports the ManageWORKS design requirement that the user be able to manage more than one entity at a time. The ManageWORKS product supports, in effect, threads of execution to allow multiple OMMs to be active simultaneously. Support for the design principle of managing many entities as easily as one is not a function of the ManageWORKS software but of the OMMs, since OMMs control the methods used to manage entities.

The Software System Design of ManageWORKS

The focus of the paper now shifts to the ManageWORKS internals that support the design principles and user interface just described.

The Application Framework

As an application, the ManageWORKS product is merely a software framework for integrating its top-level user interface with the user interfaces of the OMMs and ONMs. The ManageWORKS application consists of two main components: (1) the user interface shell and (2) the dispatcher. Figure 3 depicts the relationship between these ManageWORKS components and the OMMs and ONMs.

The user interface shell is a standard Microsoft Windows application that supports the top-level Windows user interface components—the main application window and its menu bar, tool ribbon, and status bar. The user interface shell translates all user interaction by means of the menus, tool ribbon, and mouse actions into OMM and ONM application programming interfaces (APIs) to perform work for the end user. The shell is also responsible for initializing and terminating the application, including the dispatcher.

The dispatcher is responsible for maintaining a link between the user interface shell and all the OMMs, as well as for providing service routines. The dispatcher loads and initializes all OMMs

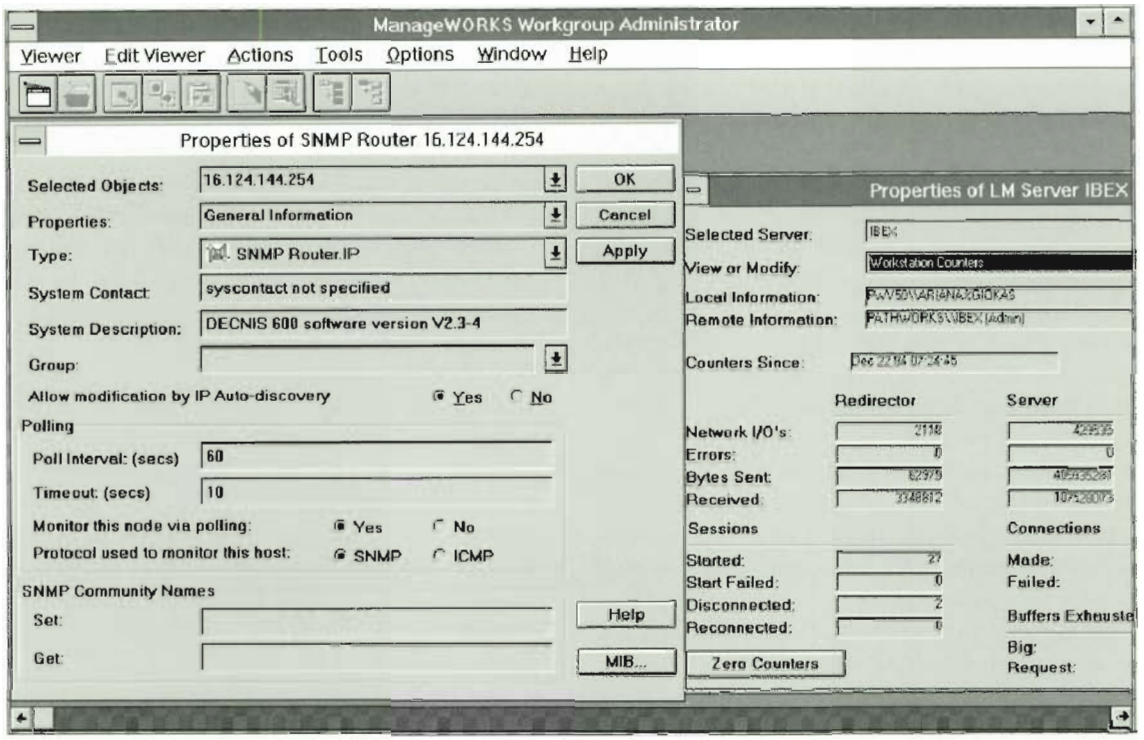


Figure 2 ManageWORKS OMM Properties Dialog Boxes

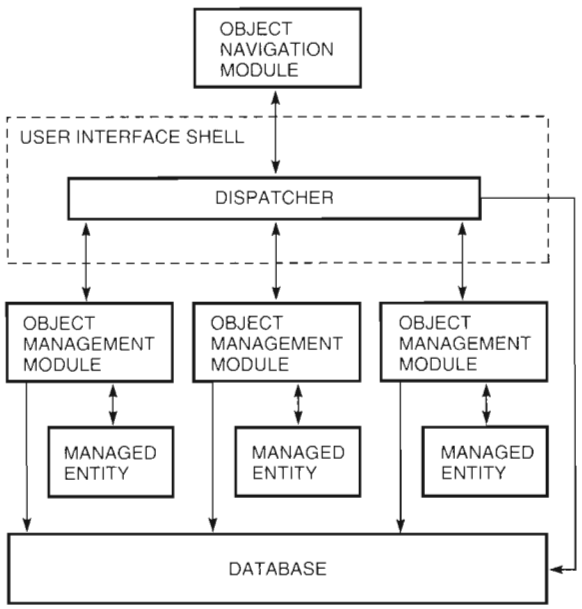


Figure 3 ManageWORKS Application Architecture

present based on an initialization file that the end user configures at installation time (or, if subsequent modules are added, by means of the Management Module Setup program). To enable this routing to occur, the dispatcher maintains a list of all OMMs loaded and the object classes that they support.

One service that the dispatcher provides for OMMs and ONMs is the ability to modify the menu bar. OMMs and ONMs may add and set menu items but only through the APIs. The ManageWORKS software ultimately controls what gets displayed in the menus based on what objects are selected in a viewer, which prevents the modules from directly manipulating the menu bar.

The Application Programming Interfaces

Once we had defined the concepts of the ManageWORKS user interface and object classes, we designed a common set of APIs that all OMM and ONM developers would employ. The APIs that emerged focused primarily on the object—both its class and its instance. Because the current set of object-oriented languages and tools does not map well to

the services supplied by the Windows system, these APIs are in a more conventional C/Pascal programming language style rather than in a C++ style.

The APIs that an OMM must support fall into three categories based upon their scope of operation: (1) module based, (2) class based, and (3) object based. All APIs have parameters that contain information pertinent to the API call, including the object identifier (OID), which identifies the object on which to perform the operation.

Module-based APIs perform initialization, termination, and information reporting for the entire OMM. The initialization includes determining how many object classes an OMM supports. This function is important because an OMM can support more than one class, e.g., a hierarchy of classes. By checking for software dependencies on the operating system or support libraries, the OMM can also make sure that the computer environment is capable of supporting the OMM. For example, Digital's implementation of the OMM that manages NetWare servers requires that the NetWare client be installed and configured on the PC. Module termination occurs before the ManageWORKS software terminates, which allows OMMs to clean up any resources they may have used. The information function provides information such as the module's name and copyright information.

Class-based APIs support the actions that apply to all objects within a class. These functions include initialization, termination, configuration, and reporting information about what actions and properties can be accessed by the end user in the ManageWORKS user interface. A class-based configuration API presents a configuration window for each class to the user; the user can then change the behavior of the object class. For example, the user can indicate whether or not files on a disk with hidden or system attributes or hidden LAN Manager file services should be displayed.

Object-based APIs provide the ability to manipulate individual objects within the ManageWORKS software. With these APIs, OMMs can accomplish all the base actions and those operations provided for in the user interface. These APIs include functions to create, delete, insert, remove, copy, get and set properties, display a properties dialog box, maintain containership relationships (e.g., technology-based hierarchies), and maintain classes that can be created and inserted into an object. Approximately 30 APIs (a small manageable set) must be implemented to be ManageWORKS compliant.

Each class- or object-based API requires an OID or list of OIDs on which to perform the operation. When called, each class API acts on a single object class. The caller manages all memory needed for the successful completion of an API, i.e., no API returns a pointer to data. APIs that can return a variable amount of information use a two-step calling convention. The first call determines the buffer size required to hold all the data; the second call retrieves that data. This two-call approach requires OMMs to efficiently gather information using OMM-specific information caches to store information retrieved from the managed entity.

ONMs contain all the module-, class-, and object-based APIs that exist in a standard OMM but also contain some viewer-specific APIs. These APIs include functions to display viewers, select displayed objects, expand objects, update objects, and retrieve displayed objects. New ONMs can be developed using these APIs.

The Object Identifier

To represent objects within the ManageWORKS software, we chose the approach of assigning an OID to each object in the system. This number embodies the information of the class to which the object belongs as well as the uniqueness of the individual instance of an object within the class.

The assignment of an OID to an object is the responsibility of the OMM. The ManageWORKS software dynamically assigns to an object class an OID that represents the class, and the OMM is responsible for creating the unique instance values within the context of that class. This approach allows OMMs the flexibility of using any strategy to assign these values, e.g., sequential assignment or mapping to a particular technology, such as an external database record.

Each OID is a 32-bit number; the high 12 bits contain information that identifies the class to which the object belongs. This bit arrangement places a limit, $2^{12}-1$, i.e., 4095 (a value of 0 is invalid), on the number of classes that can be active with ManageWORKS at any one time. The low 20 bits provide the uniqueness for each object instance within the class, providing for up to $2^{20}-1$, i.e., more than 1 million, individual instances within a single class. The advantages to using an OID lie in allowing objects to store information in any format they wish and using access functions to get at that information in a consistent manner.

Storing Information about Objects

Although the OMMs are responsible for assigning OIDs to objects within a class and for storing information about each object that can be managed, we did not want every OMM under development to have to create its own mechanism to accomplish these tasks. We decided to create an object database that would store information about objects and generate new OIDs for the OMMs.

Initial designs of this object database were to support multiple users and thus allow the sharing of information between multiple ManageWORKS users and other applications. Because the schedule for the first release of the ManageWORKS software did not give us ample time to employ a commercially available database, we decided to create our own database to support the management of object classes and object instances. This database supports only a single user and consists of indexed files for (1) object information, (2) class information, and (3) containership information. The existence of these files is hidden under a database API, which supports all the management aspects of objects, from creating and deleting classes and objects to reading and modifying attributes of those objects.

To allow future changes in the underlying technology of the database, we placed the database code into a DLL. For the second release, we created a new database DLL, with the same APIs, that works with Borland's dBase IV database implementation. By simply replacing the database DLL, all OMMs can now take advantage of having information shared between ManageWORKS users across the network. This design allows for comanagement of the LAN by multiple network administrators who have the same information available. The OMMs do not have to make any source code changes to work with this new database DLL, but additional APIs are present to allow for the use of advanced database features.

Before an OMM can create objects in the database, the object class itself must be created in the database. Because it dynamically assigns OIDs, the object database must store unique information about the class along with the OID. Each OMM must register an object class, where each class has a name that can be presented to the user in the user interface, and a class tag. The class tag is a 64-byte character string that must be unique among all OMMs. The database dynamically assigns an OID to a newly created class and maintains that mapping to the class tag. We decided that using a unique 64-byte character string would result in less conflict among OMM developers than assigning hard-coded OID values to each customer that wanted to develop an OMM. By not hard-coding the values, we ensured that each newly created object class would receive the next OID value. Thus, different end users who are using different sets of OMMs may have different OID values assigned to each of the object classes.

OMMs can use this object database to create object classes or objects within those classes, and to store any amount of information with each object. Most objects store enough information to get to another data source, thereby preventing information in the database from becoming inconsistent with the managed entity. For example, a NetWare Server OMM saves only the server name in the database because with that name the OMM can make NetWare API calls to retrieve other information.

When the object database creates an object, it assigns the object an OID within the space of that object class. Thus, OMMs can rely on the database for creating unique OIDs for each object in the system.

Another feature of the object database is the concept of transient and permanent objects. The object database DLL writes transient objects not into the database files but rather to global system memory in the Windows operating system. Having the objects in memory creates a large performance gain and avoids the problems associated with disk thrashing. To indicate the type of object that is created, the object database reserves bit 19 of the OID to use as a flag. If the bit is set by the OMM or ONM, the object is transient. When an object is created in the database, the OID for the class is passed to the database DLL with or without bit 19 set, thus determining whether the object is transient or permanent.

In our initial development work, we quickly discovered that creating all the OID entries in a database file diminished performance. This problem was most evident in the development of the DOS file system OMM. This OMM enumerates directories, which causes a disk seek operation and a disk read operation for the enumeration. Next a write of the object to the database file on the same disk causes another disk seek/write operation. This resulted in tremendous disk thrashing. We envisioned that many OMMs would enumerate and create a list of contained objects each time an object is expanded, so we wanted this operation to be fast and efficient.

Introducing New OMMs and ONMs into the ManageWORKS Software

In traditional software development, the addition of new functionality into an application generally requires source code modification and recompilation. Clearly, this approach would not allow ManageWORKS developers to meet the goal of providing an extensible application framework. Developers needed a way to write software that could become part of the ManageWORKS application without requiring changes to the application.

Since the ManageWORKS software runs in the Microsoft Windows operating system environment, software developers were able to take advantage of many features of the Windows system. We used DLLs to provide an extensible framework for the ManageWORKS product.

By creating a DLL that conforms to the set of APIs needed to manage an object or to implement a viewer, we can add new DLLs at any time to add functionality to the ManageWORKS software. Therefore, all OMMs and ONMs must be implemented as DLLs. The registration process needed to be simple and dynamic for these DLLs. Using a Windows application initialization (INI) file, the dispatcher reads the list of entries in the file and attempts to load and initialize all OMMs and ONMs defined. End users can add new OMMs by running the ManageWORKS Management Module Setup program, which simplifies the installation of any OMMs provided by either Digital or a third-party vendor.

When an OMM is introduced, the ManageWORKS software needs to assign an OID to each object class that the OMM handles. This is accomplished by asking the dispatcher for an OID for the class based upon a supplied class tag. The dispatcher then uses the object database to have the OID assigned. The dispatcher's use of the object database ensures that the OID for the class is unique to that class. OMMs can ask the object database directly, but this is merely a side effect of the dispatcher's use of the object database and is not recommended.

Interactions between ManageWORKS Components

Most ManageWORKS events occur when the user interacts with the user interface, although OMMs and ONMs can generate events that cause communication to occur between the components of the system. The usual flow of control through the ManageWORKS software begins with a viewer, the set of selected objects in a viewer, and the valid managed entity actions in the Action menu. The application uses the dispatcher to call a particular API to the correct OMM for the class of object being operated upon. In this section, we walk through three typical user interaction scenarios. For each scenario, we describe key elements of control flow between the user interface shell, the dispatcher, the ONM involved, and the OMM involved. These scenarios illustrate how the ManageWORKS elements fit and work together to achieve our primary objective, i.e., to design a user interface framework with consistent mechanisms to display, organize, and navigate through management entities for the purpose of managing one or more of those entities.

Scenario 1 This scenario outlines the process of displaying the properties dialog box of the selected object(s) in a viewer.

1. The user has selected one or more objects of the same class in a viewer by clicking with the mouse.
2. The user then chooses the Properties menu item from the Actions menu. As a reminder, this action invokes the properties dialog box, which by style guide convention, supports the viewing and modification of a managed entity's properties.
3. The ManageWORKS software queries the selected viewer for the list of selected objects and obtains the OIDs of the objects from the viewer.
4. The ManageWORKS dispatcher decodes the object class portion of the OID.
5. The ManageWORKS software tells the OMM of that object class to display the properties dialog box for the list of objects (OIDs) supplied.
6. The OMM displays a properties dialog box that contains all the supplied objects. The OMM has complete control of the user interface for this window and complete control over the access to the managed entity mechanism to get and set the properties from the managed entities.

Scenario 2 This scenario outlines the process of expanding a selected set of objects in a hierarchical viewer. Expanding an object results in the display of the object's descendants within the hierarchy defined by the OMM. The user may render this display in a hierarchical fashion with one of the hierarchical view styles or as a descendant portion of a topological view.

1. The user has selected one or more objects in a viewer by clicking with the mouse. The objects may be of the same class or of different classes.
2. The user then chooses the Expand menu item from the Actions menu.
3. The ManageWORKS software queries the selected viewer for the list of selected objects and obtains the OIDs of the objects from the viewer.
4. The ManageWORKS software tells the selected viewer to expand the list of objects supplied (the selected objects from the last call).
5. For each selected object to be expanded, the viewer queries the object by means of the dispatcher for the list of contained objects within that object. The dispatcher calls the OMM that supports the object to get the list of contained objects. The viewer repeats this process for all OIDs to be expanded.
6. For a hierarchical view, the viewer places the list of objects into the viewer in a hierarchical fashion. For a topological map view, the viewer either creates a new window or replaces the current window, depending on the choice the user has indicated through the customization dialog box. The window shows the descendant set of objects with their topological relationships.
7. For each of the contained objects, the viewer queries the object's OMM by means of the dispatcher for its name and bitmap, and to determine whether it can potentially be expanded by the user. The viewer repeats this process for each contained object to be displayed and then renders each item.

Scenario 3 This scenario outlines the process of dragging and dropping an object onto another object in a viewer. The OMM of the target object controls the semantics of this operation.

1. The viewer controls drag-and-drop operations.
2. The viewer determines the OIDs of the object(s) that the user is dragging.
3. As the user moves the mouse, the viewer receives mouse move messages from the Windows system and determines if the mouse is over a viewer. The window messages are sent directly to the viewer window.
4. If it is over a viewer, the mouse tells the target viewer what objects the user is dragging over it. The source ONM sends a ManageWORKS-defined Windows message to the target viewer window with the list of OIDs being dragged.
5. The target viewer determines what object the mouse is over and if that object is selected. The set of objects targeted to receive the dropped object comprises either the individual object, or if selected, all the selected objects in the viewer.
6. The target viewer queries the OMM of each target object about what class of object can be dropped on it. If all the target objects can accept the dragged objects, the cursor changes shape to reflect a potentially successful drop. Otherwise, the cursor changes to reflect that the drop would not succeed at this mouse location.
7. When the user drops the objects, the same verification occurs as during the drag operation. If the drop is not going to be successful, the viewer that initiated the drag operation returns the mouse cursor to the original location.
8. If the drop operation passes the verification step, each object that the user is dragging is copied by the OMM to each target object. This is done iteratively for each dragged object, and each copy has the potential for failure. For example, a DOS file can be dragged to a DOS disk class object, but when the copy is attempted, the disk may not have enough free space to successfully copy the file. When each dragged object is copied, the OMM of the target object is told that it should now contain the new object. This causes the hierarchy to be properly updated. A drag-and-drop operation that is intended to move an object is implemented as a copy followed by a removal of the original.

Conclusions

We feel that we have been successful at building a unique user interface framework that integrates a diverse set of applications; the design essentially meets all but one of the objectives we established. Because by design we limited the scope of services provided by the framework, we could not meet all of our end-user objectives. Specifically, the responsibility of allowing the user to manage many entities as though they were one fell on the OMMs and not on the framework itself. Although we would have liked the framework to provide this service,

such a design was not feasible, given that the OMM controlled both the access to the managed entity and the user interface to view and modify entity properties.

The reader should observe that the first two major releases of the ManageWORKS software provide few core services. The core services include the user interface shell, the viewers, and the object database that ship with the ManageWORKS product and the ManageWORKS Software Developer's Kit. These components serve as a unifying framework for the functional modules, which provide the user with tools to manage entities and are thus the "heart and soul" of the environment. Future development of core framework services is under consideration. Among the areas under active consideration are Windows Object Linking and Embedding (OLE) support and scripting support for inter- and intra-OMM control. Such services would make ONMs and OMMs more consistent, useful, and powerful for the end user. At the same time, these services would free the individual developer from writing this code and thus provide the developer the freedom to focus on the value-added functionality.

Acknowledgments

Many people contributed a great deal to the design and implementation of the ManageWORKS product. Although the contributors are too numerous to mention individually, we would like to acknowledge the functional groups within the PATHWORKS organization to which they belong, namely, Business Management, Marketing, Human Factors Engineering, Systems Quality Engineering, Documentation, Release Engineering, Field Test Administration, and, of course, Software Development Engineering.

References

1. K. Holtzblat and S. Jones, "Contextual Inquiry: A Participatory Technique for System Design" in *Participatory Design: Principles and Practice,* A. Namioka and D. Schuler, eds. (Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., 1993).
2. *Microsoft Windows Guide to Programming* (Redmond, WA: Microsoft Press, 1990).
3. Windows 3.1 Software Developer's Kit, Control Panel Applets in Online Help (Redmond, WA: Microsoft Press, 1992).
4. C. Strutt and D. Shurtleff, "Architecture for an Integrated, Extensible Enterprise Management Director" in *Integrated Network Management,* vol. 1, B. Meandzja and J. Westcott, eds. (Amsterdam: North-Holland, Elsevier, 1989): 61–72.
5. *ManageWORKS Programming Guide* (Maynard, MA: Digital Equipment Corporation, Order No. AA-QADFB-TE, 1994).
6. *ManageWORKS Programmer's Reference* (Maynard, MA: Digital Equipment Corporation, Order No. AA-QADGB-TE, 1994).

James E. Johnson

The Structure of the OpenVMS Management Station

The OpenVMS Management Station software provides a robust client-server application between a PC running the Microsoft Windows operating system and several OpenVMS cluster systems. The initial version of the OpenVMS Management Station software concentrated on allowing customers to handle the system management functionality associated with user account management. To achieve these attributes, the OpenVMS Management Station software uses the data-sharing aspects of OpenVMS cluster systems, a communications design that is secure and that scales well with additional target systems, and a management display that is geared for the simultaneous management of multiple similar systems.

Overview

The OpenVMS Management Station version 1.0 software provides a robust, scalable, and secure client-server application between a personal computer (PC) running the Microsoft Windows operating system and several OpenVMS systems. This management tool was developed to solve some very specific problems concerning the management of multiple systems. At the same time, the project engineers strove for a release cycle that could bring timely relief to customers in installments.

Before the advent of this new software, all OpenVMS base system management tools have either executed against one system, such as AUTHORIZE, or against a set of systems in sequence, such as SYSMAN. Furthermore, the existing tools that do provide some primitive support for the management of multiple systems either do not take advantage of or do not take into account the inherent structure of a VMScluster system.

In contrast, the OpenVMS Management Station product was designed from the outset for efficient execution in a distributed, multiple system configuration. The OpenVMS Management Station tool supports parallel execution of system management requests against several target OpenVMS systems or VMScluster systems. Furthermore, the software incorporates several features that make such multiple target requests natural and easy for the system manager.

Data from customer surveys indicated the need for a quick response to the problems of managing OpenVMS systems. For this reason, the project team chose a phased delivery approach, in which a series of frequent releases would be shipped, with support for a small number of system management tasks provided in an individual release.

The initial version of the OpenVMS Management Station software concentrated on providing the system management functionality associated with user account management. This goal was achieved by using a project infrastructure that supported frequent product releases. This paper describes the OpenVMS Management Station software, concentrating on the client-server structure. It also presents the issues and trade-offs that needed to be faced for successful delivery.

Managing OpenVMS User Accounts

Managing user accounts on an OpenVMS operating system is a relatively complicated task.[1] The manner in which the user is represented to the system manager is the cause of much complexity. The attributes that define a user are not located in one place, nor is much emphasis placed on ensuring consistency between the various attributes.

For example, Table 1 gives the attributes of an OpenVMS user stored in various files, including the user authorization file (SYSUAF.DAT), the rightslist file (RIGHTSLIST.DAT), and the DECnet network

Table 1 Breakdown of Data Stores and Management Utilities for OpenVMS Users

Data Store	Attributes	Management Utility
SYSUAF.DAT	Username, Authorization data (e.g., passwords), process quotas, login device, and directory	AUTHORIZE
RIGHTSLIST.DAT	Rights identifiers	AUTHORIZE
NET$PROXY.DAT	Remote<->local user DECnet proxy mappings	AUTHORIZE
VMS$MAIL_PROFILE.DAT	User's mail profile	MAIL
QUOTA.SYS (per disk)	User's disk quota	DISKQUOTA
Login directory	User's home directory	CREATE/DIRECTORY
TNT$UADB.DAT	User's location, phone number, and organization information	<new with OpenVMS Management Station software>

proxy file (NET$PROXY.DAT). Prior to the OpenVMS Management Station product, these files were managed by a collection of low-level utilities, such as AUTHORIZE. Although these utilities provide the ability to manipulate the individual user attributes, they offer little support for ensuring that the overall collection of user attributes is consistent. For instance, none of these utilities would detect that a user's account had been created with the user's home directory located on a disk to which the user had no access.

All of these factors create additional complexity for an OpenVMS system manager. This complexity is compounded when a number of loosely related OpenVMS systems must be managed at various sites. The user account management features of the OpenVMS Management Station product are designed to alleviate or remove these additional complexities for the OpenVMS system manager.

OpenVMS System Configurations

The OpenVMS operating system can be used in many ways. The features of the VMScluster method allow systems to expand by incrementally adding storage or processing capacity. In addition, the OpenVMS operating system is frequently used in networked configurations. Its inherent richness leads to a large and diverse range in the possible OpenVMS configurations. The skill and effort required to manage the larger configurations is considerable.

For instance, Figure 1 shows a possible customer configuration, in which a number of VMScluster systems extend across a primary and a backup site. Each cluster has a somewhat different purpose, as given in Table 2. Here OpenVMS workstations are deployed to users who need dedicated processing power or graphics support, and personal computers are deployed in other departments for data access and storage. Finally, the table lists some groups of users who need access to multiple systems, sometimes with changed attributes. The system manager for this type of configuration would repeatedly perform many tasks across several targets, such as systems or users, with small variations from target to target. The OpenVMS Management Station product was designed to operate well in configurations such as this.

A distributed system is clearly necessary to support effective and efficient systems management for configurations such as the one shown in Figure 1. A distributed system should support parallel execution of requests, leverage the clusterwide attributes of some system management operations, and provide for wide area support. These characteristics are expanded in the remainder of this section.

Supporting Parallel Execution

Support of parallel execution has two different implications. First, the execution time should rise slowly, or preferably remain constant, as systems are added. This implies that the execution against any given target system should be overlapped by the execution against the other target systems. Second, for parallel execution to be usable in a wider range of cases, it should be easy and straightforward to make a request that will have similar, but not identical, behavior on the target systems. For instance, consider adding a user for a new member of the development staff in the configuration shown in Figure 1. The new user would be privileged on the

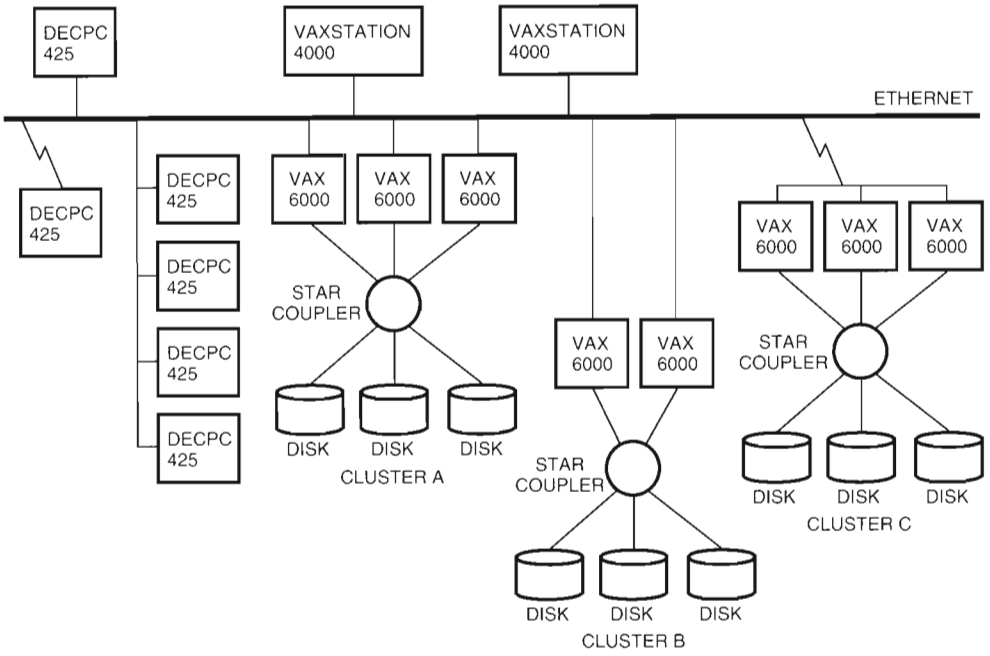


Figure 1 Distributed OpenVMS System Configuration

development VMScluster system, but unprivileged on the production cluster. It should be straightforward to express this as a single request, rather than as two disparate ones.

Leveraging VMScluster Attributes

OpenVMS system management tasks operate against some resources and attributes that are shared clusterwide, such as modifications to the user authorization file, and some that are not shared, such as the system parameter settings.

In the first case, the scope of the resource extends throughout the VMScluster system. Here, it is desirable (and when the operation is not idempotent, it is necessary) for the operation to execute once within the VMScluster system. In the latter case, the operation must execute against every system within the cluster that the system manager wants to affect. Also, the set of resources that falls into the first case or the second is not fixed. In the OpenVMS operating system releases, the ongoing trend is to share resources that were node-specific

Table 2 Usage and User Community for Sample Configuration

Name	Usage	User Community
A	Main production cluster	Operations group Production group Development group (unprivileged)
B	Development cluster	Operations group Development group (full development privileges)
C	Backup production cluster and main accounting cluster	Operations group Development group (unprivileged) Production group Accounting group
	Workstations	Workstation owner Some of operations group

throughout a VMScluster system. The OpenVMS Management Station software must handle resources that have different scopes on different systems that it is managing at the same time.

Wide Area Support

Management of a group of OpenVMS systems is not necessarily limited to one site or to one local area network (LAN). Frequently there are remote backup systems, or the development site is remote from the production site. Almost certainly, the system manager needs to be able to perform some management tasks remotely (from home). Therefore, any solution must be able to operate outside of the LAN environment. It should also be able to function reasonably in bandwidth-limited networks, regardless of whether or not the slower speed lines are to a few remote systems, or between the system manager and all the managed systems.

OpenVMS Management Station Structure

The resulting structure for the OpenVMS Management Station software is shown in Figure 2. The components contained within the dashed box are present in the final version 1.0 product. The other components were specified in the design, but were unnecessary for the initial release.

The client software on the PC uses the ManageWORKS management framework from Digital's PATHWORKS product. This extensible framework provides hierarchical navigation and presentation support, as well as a local configuration database.[2] The framework dispatches to Object Management Modules (OMMs) to manage individual objects. OpenVMS Management Station has three OMMs that are used to organize the system manager's view of the managed systems. These are Management Domains, VMScluster Systems, and OpenVMS Nodes. In addition, two OMMs manage user accounts: OpenVMS Accounts and OpenVMS User. The first OMM is used to retrieve the user accounts and to create subordinate OpenVMS User objects in the ManageWORKS framework hierarchy. The second contains the client portion of the OpenVMS user account management support. Underlying the last two OMMs is the client communications layer. This provides authenticated communications to a server.

The server software on the OpenVMS systems consists of a message-dispatching mechanism and a collection of server OMMs that enact the various management requests. The dispatcher is also

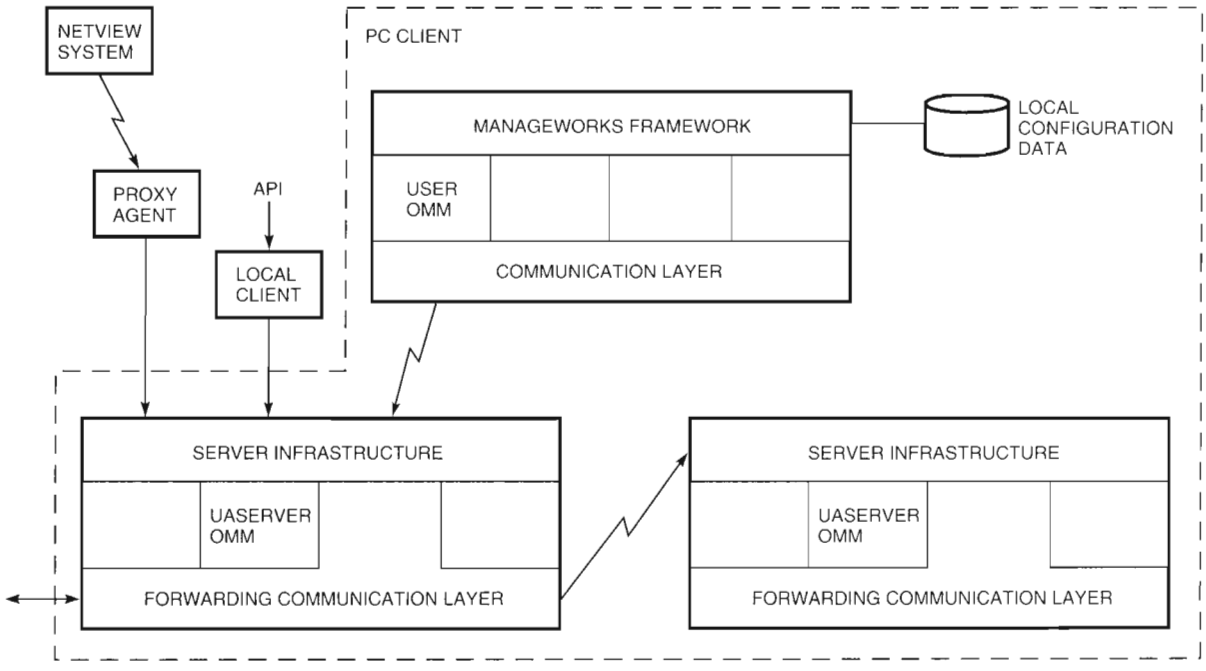


Figure 2 OpenVMS Management Station Structure

responsible for forwarding the management request to all target VMScluster systems and independent systems, and for gathering the responses and returning them to the client. The version 1.0 server contains two OMMs: UAServer and Spook. The former implements the server support for both the OpenVMS Accounts and OpenVMS User OMMs. The Spook OMM implements the server component of the authentication protocol.

Other clients were not built for version 1.0 but were planned into the design. Specifically, these items are (1) a local client to provide a local application programming interface (API) to the functions in the server, and (2) a proxy agent to provide a mapping between Simple Network Management Protocol (SNMP) requests and server functions.

Design Alternatives

Before this structure was accepted, the designers considered a number of alternatives. The two areas with many variables to consider were the placement of the communications layer and the use of a management protocol.

Communications Layer Placement The first major structural question concerned the placement of the communications layer in the overall application.

At one extreme, the client could have been a display engine, with all the application knowledge in the servers. This design is similar to the approach used for the X Window System and is sufficient for the degenerate case of a single managed system. Without application knowledge in the client, however, there was no opportunity for reduction of data, or for the simplification of its display, when attempting to perform management requests to several target systems.

At the other extreme, the application knowledge could have been wholly contained within the client. The server systems would have provided simple file or disk services, such as Distributed Computing Environment (DCE) distributed file server (DFS) or Sun's Network File Service (NFS). Since application knowledge would be in the client, these services would provide management requests to either a single managed system or to multiple systems. However, they scale poorly. For instance, in the case of user account management, seven active file service connections would be required for each managed system! Furthermore, these services exhibit very poor responsiveness if the system manager is remotely located across slower speed lines from the managed systems. Finally, they require that the client understand the scope of a management resource for all possible target OpenVMS systems that it may ever manage.

These various difficulties led the project team to place the data gathering, reduction, and display logic in the client. The client communicates to one of the managed systems, which then forwards the requests to all affected independent systems or VMScluster systems. Similarly, replies are passed through the forwarding system and sent back to the client. The chosen system is one that the system manager has determined is a reasonable choice as a forwarding hub.

Note that the forwarding system sends a request to one system in a VMScluster. That system must determine if the request concerns actions that occur throughout the VMScluster or if the request needs to be forwarded further within the VMScluster. In the second case, that node then acts as an intermediate forwarding system.

This structure allows the client to scale reasonably with increasing numbers of managed systems. The number of active communication links is constant, although the amount of data that is transferred on the replies increases with the number of targeted managed systems. The amount of local state information increases similarly. Although it is not a general routing system, its responsiveness is affected less by either a system manager remote from all the managed systems, or by the management of a few systems at a backup site. Finally, it allows the managed VMScluster system to determine which management requests do or do not need to be propagated to each individual node.

Use of Standard Protocols The second major structural question concerned the use of de facto or de jure standard enterprise management protocols, such as SNMP or Common Management Information Protocol (CMIP).[3,4] Both protocols are sufficient to name the various management objects and to encode their attributes. Neither can direct a request to multiple managed systems. Also, neither can handle the complexities of determining if management operations are inherently clusterwide or not. The project team could have worked around the shortcomings by using additional logic within the management objects. This alternative would have reduced the management software's use of either protocol to little more than a message encoding

scheme. However, it was not clear that the result would have been useful and manageable to clients of other management systems, such as NetView.

On a purely pragmatic level, an SNMP engine was not present on the OpenVMS operating system. The CMIP-based extensible agent that was available exceeded the management software's limits for resource consumption and responsiveness. For instance, with responsiveness, a simple operation using AUTHORIZE, such as "show account attributes," typically takes a second to list the first user account and is then limited by display bandwidth. For successful adoption by system managers, the project team felt that any operation needed to be close to that level of responsiveness. Early tests using the CMIP-based common agent showed response times for equivalent operations varied from 10 to 30 seconds before the first user was displayed. Remaining user accounts were also displayed more slowly, but not as noticeably.

In the final analysis, the project engineers could have either ported an SNMP engine or corrected the resource and responsiveness issues with the CMIP-based common agent. However, either choice would have required diverting considerable project resources for questionable payback. As a result, the product developers chose to use a simple, private request-response protocol, encoding the management object attributes as type-length-value sequences (TLVs).

Client Component

With the OpenVMS Management Station, the client is the component that directly interacts with the system manager. As such, it is primarily responsible for structuring the display of management information and for gathering input to update such management information. This specifically includes capabilities for grouping the various OpenVMS systems according to the needs of the system manager, for participating in the authentication protocol, and for displaying and updating user account information.

Grouping OpenVMS Systems for Management Operations

The system manager is able to group individual systems and VMScluster systems into loose associations called domains. These domains themselves may be grouped together to produce a hierarchy. The system manager uses hierarchies to indicate the targets for a request.

Note that these hierarchies do not imply any form of close coupling. Their only purpose is to aid the system manager in organization. Several different hierarchies may be used. For a given set of systems, a system manager may have one hierarchy that reflects physical location and another that reflects organization boundaries.

Figure 3 shows a typical hierarchy. In the figure, the system manager has grouped the VMScluster systems, PSWAPM and PCAPT, into a domain called My Management Domain. The display also shows the results of a "list users" request at the domain level of the hierarchy. A "list users" request can also be executed against a single system. For instance, to obtain the list of users on the PCAPT VMScluster system, the system manager need only expand the "OpenVMS Accounts" item directly below it.

Participation in the Authentication Protocol

It was an essential requirement from the start for the OpenVMS Management Station software to be at least as secure as the traditional OpenVMS system management tools. Furthermore, due to the relatively insecure nature of PCs, the product could not safely store sensitive data on the client system. For usability, however, the product had to limit the amount and frequency of authentication data the system manager needed to present.

As a result, two OMMs, the VMScluster and the OpenVMS Node, store the OpenVMS username that the system manager wishes to use when accessing those systems. For a given session within the ManageWORKS software, the first communication attempt to the managed system results in a request for a password for that username. Once the password is entered, the client and the server perform a challenge-response protocol. The protocol establishes that both the client and the server know the same password without exchanging it in plain text across the network. Only after this authentication exchange has successfully completed, does the server process any management requests.

The hashed password is stored in memory at the client and used for two further purposes. First, if the server connection fails, the client attempts to silently reconnect at the next request (if a request is outstanding when the failure occurs, that request reports a failure). This reconnection attempt also undergoes the same authentication exchange. If the hashed password is still valid, however, the reconnection is made without apparent interruption or

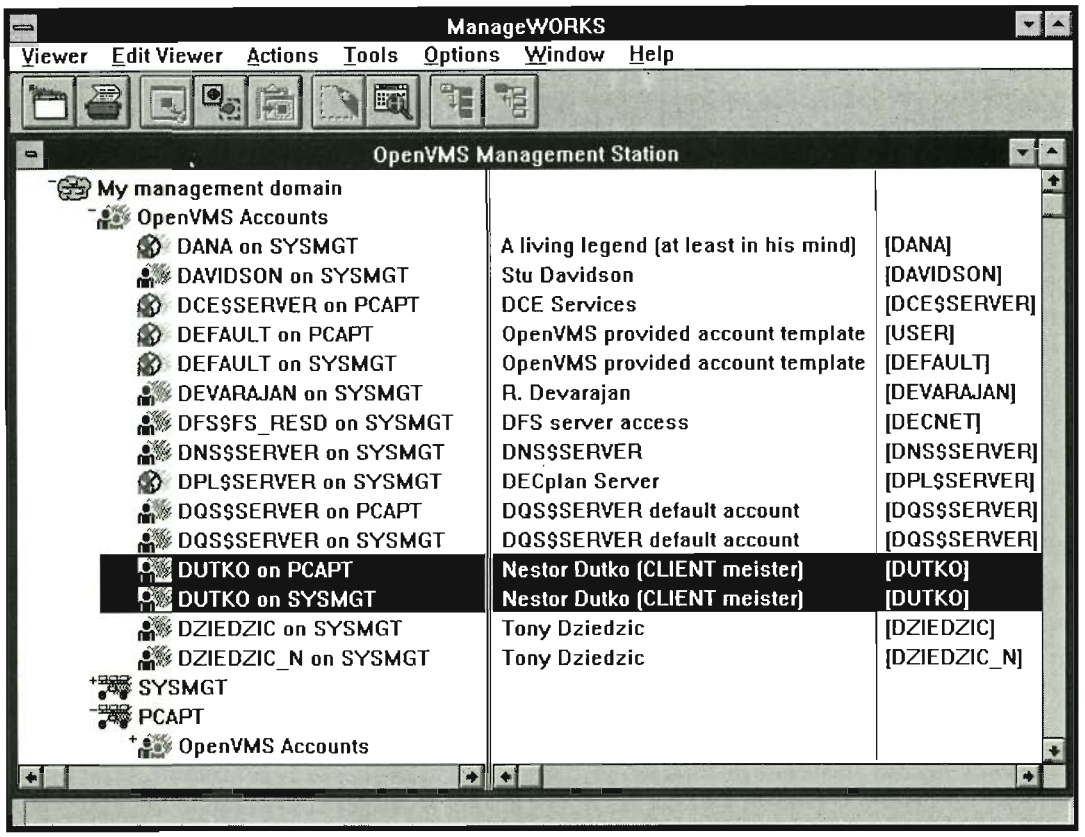


Figure 3 Management Domain View

requests for input from the system manager. Second, the hashed password is used as a key to encrypt particularly sensitive data, such as new passwords for user accounts, prior to their transmission to the server.

The resulting level of security is quite high. It certainly exceeds the common practice of remotely logging in to OpenVMS systems to manage them.

Display and Update of User Account Information

The OpenVMS Management Station version 1.0 client software primarily supports user account management. This support is largely contained in the OpenVMS User OMM. This module presents the OpenVMS user account attributes in a consistent, unified view.

The main view from the OpenVMS User OMM is called the zoom display. This series of related windows displays and allows modification to the user account attributes. The displays are organized so that related attributes appear in the same window. For instance, all the mail profile information is in one window.

The first window to be displayed is the characteristics display, which is shown in Figure 4. This window contains general information about the user that was found during usability testing to be needed frequently by the system manager. Occasionally, information was needed in places that did not match its internal structure. For instance, the "new mail count" was found to have two windows: the user flags display, which had the login display attributes, and the mail profile display.

The OpenVMS User OMM and the zoom display organize the attributes into logical groupings, simplify the display and modification of those attributes, and provide fairly basic attribute consistency enforcement. The project team did encounter one case in which no standard text display proved sufficiently usable. This was in the area of access time restrictions. All attempts to list the access times as text proved too confusing during usability testing. As a result, the project developers produced

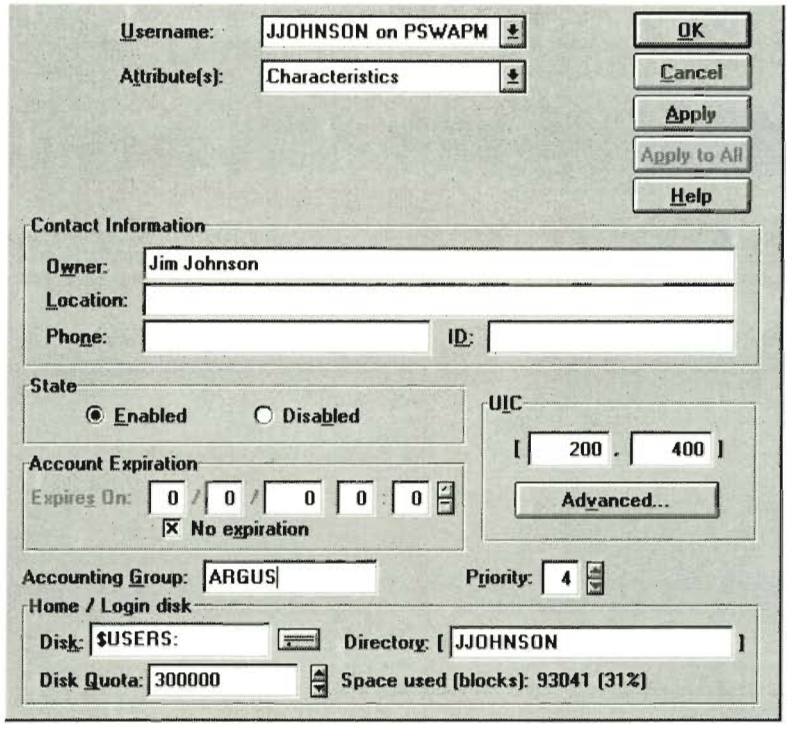


Figure 4 User Characteristics Display

a specialized screen control that displayed the time range directly, as shown in the Time Restrictions section of Figure 5. Later, system managers who participated in the usability testing found this to be very usable.

The display and presentation work for the OpenVMS User OMM was necessary for usability. However, this does not directly address the need to support requests against multiple simultaneous targets. For the OpenVMS User OMM, these targets may be either multiple VMScluster systems or independent systems, multiple users, or a combination of either configuration with multiple users.

At its simplest, this support consisted of simply triggering a request to have multiple targets. This is done through the Apply to All button on any of the zoom windows. By pressing this button, the system manager directs the updates to be sent to all user accounts on all target systems listed in the user name field. This action is sufficient if the system manager is performing a straightforward task, such as "set these users' accounts to disabled." It is not sufficient in a number of cases.

For example, one interesting case involves user account resource quotas. One reason a system manager changes these settings is to accommodate a new version of an application that needs increased values to function correctly. Prior to the development of the OpenVMS Management Station tool, the system manager had to locate all the users of this application, examine each account, and increase the resource quotas if they were below the application's needs. Conversely, with the OpenVMS Management Station product, the system manager selects the users of the application in the domain display (Figure 3), and requests the zoom display for the entire set. The system manager then proceeds to the user quota display and selects the quotas to change. The selection takes the form of a conditional request—in this case an At Least condition—and the value to set. The system manager then presses the Apply to All button, and the changes are carried out for all selected users. Figure 6 shows the user quota display.

Communications Component

The communications component is responsible for managing communications between the client and servers. It provides support for transport-independent, request-response communications, automated

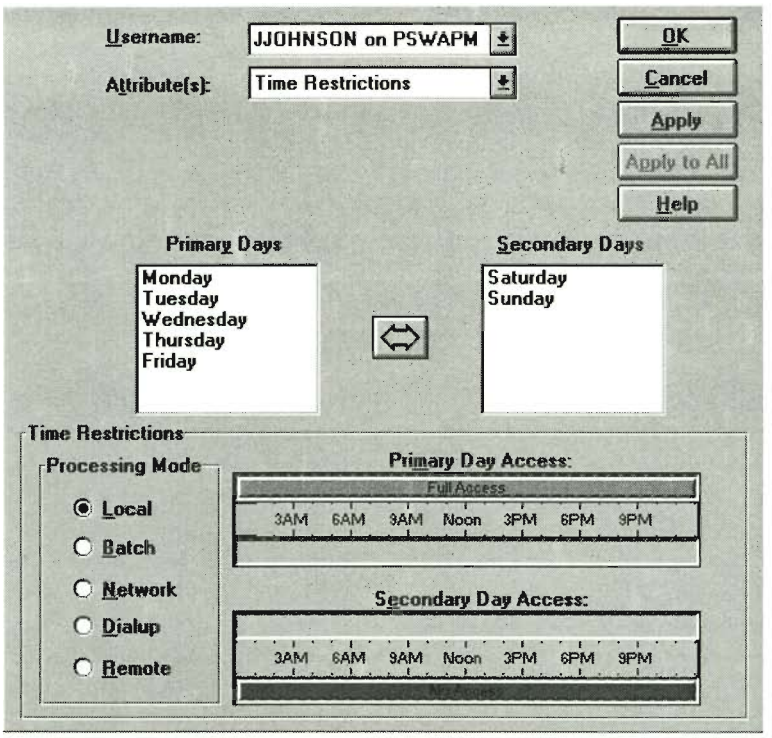


Figure 5 User Time Restrictions Display

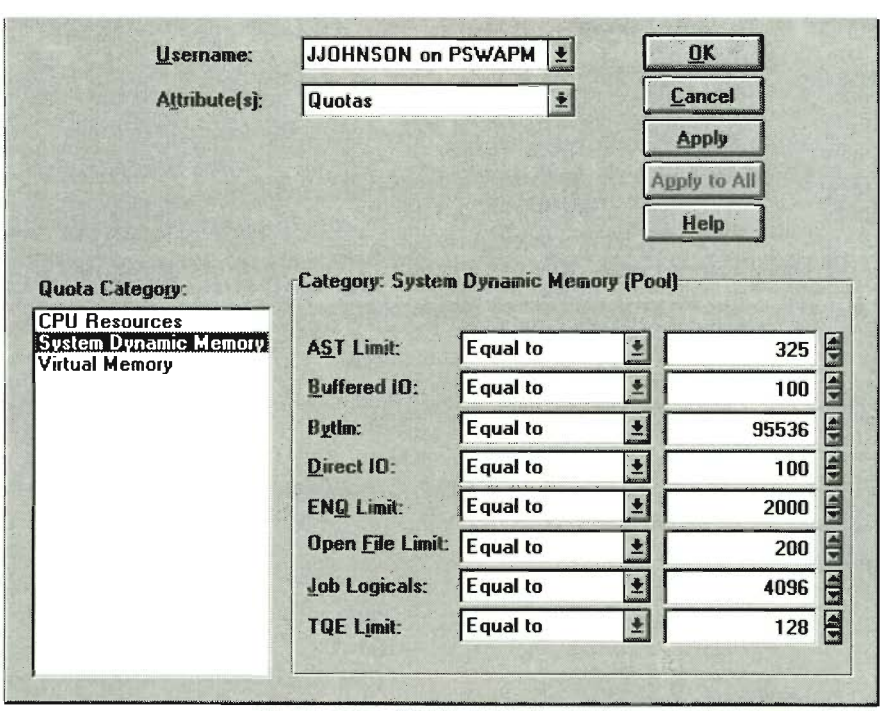


Figure 6 User Quota Display

reconnection on failure, and support routines for formatting and decoding attributes in messages.

Because of the request-response nature of the communications, the project team's first approach was to use remote procedure calls for communications, using DCE's remote procedure call (RPC) mechanism.[5] This matches the message traffic for the degenerate case of a single managed system. Management of multiple systems can easily be modeled by adding a continuation routine for any given management service. This routine returns the next response, or a "no more" indication.

The RPC mechanism also handles much of the basic data type encoding and decoding. A form of version support allows the services to evolve over time and still interoperate with previous versions.

The project team's eventual decision not to use DCE's RPC was not due to technical concerns. The technology was, and is, a good match for the needs of the OpenVMS Management Station software. Instead, the decision was prompted by concerns for system cost and project risk. At the time, both the OpenVMS Management Station product and the OpenVMS DCE port were under development. The DCE on OpenVMS product has since been delivered, and many of the system cost concerns, such as the license fees for the RPC run time and the need for non-OpenVMS name and security server systems, have been corrected.

In the end, the OpenVMS Management Station software contained a communications layer that hid many of the details of the underlying implementation, offering a simple request-response paradigm. The only difference with an RPC-style model is that the data encoding and decoding operations were moved into support routines called directly by the sender or receiver, rather than by the communications layer itself. In future versions, the goal for this layer is to support additional transports, such as simple Transmission Control Protocol/Internet Protocol (TCP/IP) messages or DCE's RPC. An investigation into providing additional transports is currently under way.

The remainder of this section describes the communications layer in more detail, including the mechanisms provided to the client OMMs, how reconnection on failure operates, and the message encoding and decoding support routines.

Client Request-response Mechanisms

The OMMs in the client system call the communications layer directly. To make a request, an OMM first updates the collection of systems that are to receive any future management requests. Assuming this was successful, the OMM then begins the request processing by retrieving the version number for the current forwarding server. Based on that, the OMM then formats and issues the request. Once the request has been issued, the OMM periodically checks to see if either the response has arrived or the system manager has canceled the request. Upon arrival of the response, it is retrieved and the message data decoded.

To perform this messaging sequence, the OMM uses a pair of interfaces. The first is used to establish and maintain the collection of systems that are to receive any management requests. The second interface, which is compatible with X/Open's XTI standard, is used to issue the request, determine if the response is available, and to retrieve it when it is.[6] A third interface that supports the encoding and decoding of message data is described in a following section.

Reconnection on Failure

The OpenVMS Management Station product attempts to recover from communications failures with little disruption to the system manager through the use of an automated reconnection mechanism. This mechanism places constraints on the behavior of the request and response messages. Each request must be able to be issued after a reconnection. Therefore, each request is marked as either an initial request, which does not depend on server state from previous requests, or as a continuation request, which is used to retrieve the second or later responses from a multiple target request and does depend on existing server state.

If an existing communications link fails, that link is marked as unconnected. If a response were outstanding, an error would be returned instead of a response message. When the communications layer is next called to send a request across the unconnected link, an automated reconnection is attempted. This involves establishing a network connection to a target system in the request. Once the connection has been established, the authentication protocol is executed, using the previously supplied authentication data. If authentication succeeds, the request is sent. If it is a continuation request, and the target server has no existing state for that request, an error response is returned.

At most, the resulting behavior for the system manager is to return an error on a management

request, indicating that communication was lost during that request's execution. If no request was in progress, then there is no apparent disruption of service.

Message Encoding and Decoding

Messages from the OpenVMS Management Station tool are divided into three sections. The first section contains a message header that describes the length of the message, the protocol version number in use, and the name of the target OMM. The second section contains the collection of target systems for the request. The third section contains the data for the OMM. This last section forms the request and is the only section of the message that is visible to the OMMs.

The OMM data for a request is typically constructed as a command, followed by some number of attributes and command qualifiers. For instance, a request to list all known users on a system, returning their usernames and last login time, could be described as this:

```
COMMAND      LIST_USERS
MODIFIER     USERNAME = "*"
ATTRIBUTES   USERNAME,
             LAST_LOGIN_TIME
```

The OMM data for a response is typically a status code, the list of attributes from the request, and the attributes' associated values. There may be many responses for a single request. Using the LIST_USERS example from above, the responses would each look like a sequence of:

```
STATUS       SUCCESS
ATTRIBUTES   USERNAME (<value>)
             LAST_LOGIN_TIME (<value>)
```

There are many possible attributes for an OpenVMS user. To make later extensions easier and to limit the number of attributes that must be retrieved or updated by a request, the OMM data fields are self-describing. They consist of a sequence of message items that are stored as attribute code/item length/item value. The base data type of each attribute is known and fixed.

Message encoding is supported by a set of routines. The first accepts an attribute code and its associated data item. It appends the appropriate message item at the end of the current message. This is used to encode both requests and responses. The second routine takes a message buffer and an attribute code, returning the attribute's value and a status code indicating if the attribute was present in the message buffer. The client uses this routine to locate data in a response. The third routine takes a message buffer, a table listing the attribute codes that are of interest to the caller, and an action routine that is called for each message item that has an attribute code found in the table. The server OMMs use this routine to process incoming requests.

Handling of Complex Data Types

In general, the interpretation of data between the client and server systems did not pose a significant concern. There was no floating-point data, and the integer and string data types were sufficiently similar not to require special treatment. However, the OpenVMS Management Station software did need a few data types to process that were not simple atomic values. These required special processing to handle. This processing typically consisted of formatting the data type into some intermediate form that both client and server systems could deal with equally well.

For instance, one such data type is the timestamp. In the OpenVMS operating system, times are stored as 64-bit quadword values that are 100-nanosecond offsets from midnight, November 18, 1858. This is not a natural format for a Microsoft Windows client. Date and time display formats vary greatly depending on localization options, so the data needed to be formatted on the local client. The developers used an approach that decomposed the native OpenVMS time into a set of components, similar to the output from the $NUMTIM system or the UNIX tm structure. This decomposed time structure was the format used to transmit timestamp information between the client and server.

Server Component

With the OpenVMS Management Station product, the server component is responsible for enacting management requests that target its local system. The server also must forward requests to all other VMScluster systems or independent systems that any incoming request may target. The server is a multithreaded, privileged application running on the managed OpenVMS systems. It consists of an infrastructure layer that receives incoming requests and dispatches them, the server OMMs that enact the management requests for the local system, and a forwarding layer that routes management requests to other target systems and returns their responses.

Server Infrastructure

The server infrastructure, shown in Figure 7, is responsible for dispatching incoming requests to the server OMMs and the forwarding layer. It has a set of threads, one for each inbound connection, a pair of work queues that buffer individual requests and responses, and a limited set of worker threads that either call the appropriate OMM or forward the request.

The inbound connection threads are responsible for ensuring that the request identifies a known OMM and meets its message requirements. The connection threads must also ensure that the OMM version number is within an acceptable range and that the link is sufficiently authenticated. The inbound threads are then responsible for replicating the request and placing requests that have only one target system in the request work queue. Once a response appears in the response work queue, these threads return the response to the client system.

A fixed number of worker threads are responsible for taking messages from the request work queue and either forwarding them or calling the appropriate local OMM. Each result is placed in the response queue as a response message. A fixed number of five worker threads was chosen to ensure that messages with many targets could not exhaust the server's resources. Responsiveness and resource usage were acceptable throughout the development and testing phases of the project, and the number of worker threads was kept at five.

In addition to the basic thread structure, the infrastructure is responsible for participating in the authentication exchange for inbound connections. This is accomplished through the use of a specialized server OMM, called Spook. The Spook OMM uses the basic server infrastructure to ensure that authentication requests are forwarded to the appropriate target system. This mechanism reduced the amount of specialized logic needed for the authentication protocol: for this reason, the server OMMs must declare if they require an authenticated link before accepting an incoming request.

Server OMM Structure

The server OMMs are at the heart of the server. These OMMs are loaded dynamically when the server initializes.

Figure 8 shows the structure of the UAServer OMM in OpenVMS Management Station version 1.0. The server OMM consists of the main application module, the preprocessing routine, and the postprocessing routine. The interfaces are synchronous, passing OMM data sections from the request and response message buffers. In addition, the main application module executes in the security context, called a persona, of the authenticated caller. This allows normal access checking and auditing in the OpenVMS operating system to work transparently.

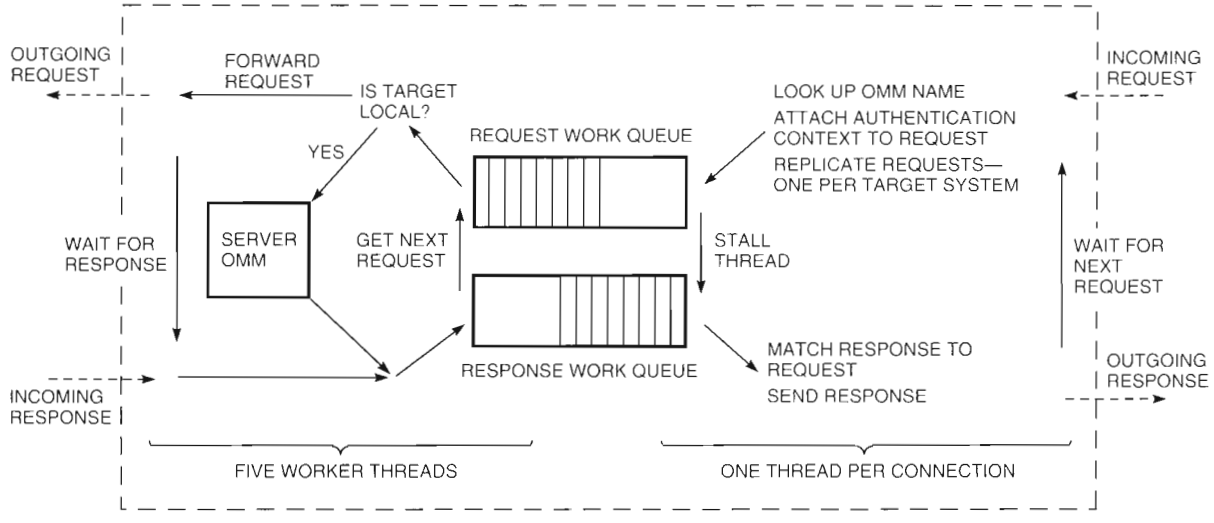


Figure 7 Server Infrastructure and Message Flow

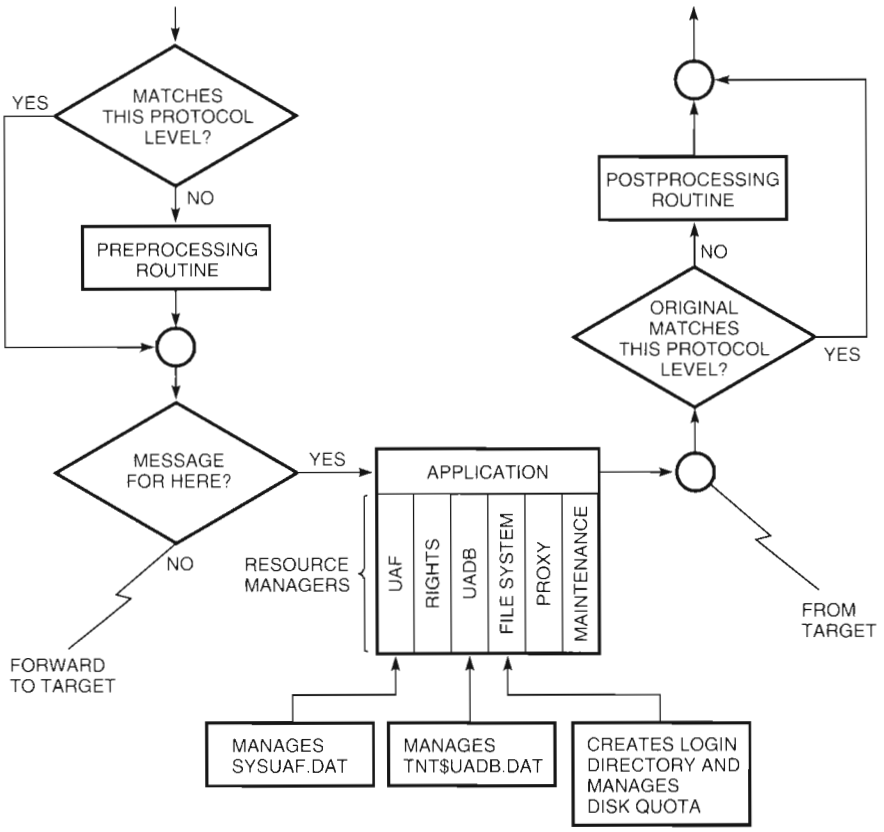


Figure 8 UAServer OMM

The preprocessing and postprocessing routines are used to ease interoperation of multiple versions. They are called if the incoming request has a different, but supported, OMM version number than the one for the local OMM. The resulting OMM data section is at the local OMM's version. These routines hide any version differences in the OMM's data items and free the main application from the need to handle out-of-version data items. If the preprocessing routine is called, the server infrastructure always calls the postprocessing routine, even if an error occurred that prevented the main OMM application from being called (for instance, by a link failure during forwarding). This allows the two routines to work in tandem, with shared state.

The actual management operations take place in the main application portion of the server OMM. It is structured with an application layer that provides the interface to the management object, such as the user account. This uses underlying resource managers that encapsulate the primitive data stores, such as the authorization file. The application layer knows what resources are affected by a given management request. Each resource manager knows how to perform requested modifications to the specific resource that it manages.

For instance, the UAServer application layer knows that the creation of a new user involves several resource managers, including the authorization file and file system resource managers. However, it does not specifically know how to perform low-level operations such as creating a home directory or modifying a disk quota entry. In comparison, the file system resource manager knows how to do these low-level operations, but it does not recognize the higher level requests, such as user creation.

The application layer for all OMMs offers an interface and a buffer. The request message passes the OMM data section to the interface, and the buffer holds the OMM data section for the response message. Similarly, all resource managers accept an OMM data section for input and output parameters, ignoring any OMM data items for attributes outside their specific resource. Because of the loose

coupling between the resource managers and the application layer, the resource managers can be easily reused by server OMMs developed later.

Summary

The OpenVMS Management Station tool has demonstrated a robust client-server solution to the management of user accounts for the OpenVMS operating system. It provides increases in functionality and data consistency over system management tools previously available on the OpenVMS operating system. In addition, the OpenVMS Management Station software is focused on the management of several loosely associated VMScluster systems and independent systems. It has addressed the issues concerning performance, usability, and functionality that arose from the need to issue management requests to execute on several target systems.

Acknowledgments

I wish to thank the Argus project team of Gary Allison, Lee Barton, George Claborn, Nestor Dutko, Tony Dziedzic, Bill Fisher, Sue George, Keith Griffin, Dana Joly, Kevin McDonough, and Connie Pawelczak for giving me a chance to work on such an interesting and exciting project. I also wish to thank Rich Marcello and Jack Fallon for providing support and encouragement to the team throughout the project, and for their further encouragement in writing about this experience.

References

1. *OpenVMS AXP Guide to System Security* (Maynard, MA: Digital Equipment Corporation, May 1993): 5-1 to 5-37.

2. D. Giokas and J. Rokicki, "The Design of ManageWORKS: A User Interface Framework," *Digital Technical Journal,* vol. 6, no. 4 (Fall 1994, this issue): 63–74.

3. J. Case, M. Fedor, M. Schoffstall, and J. Davin, *Network Working Group,* Internet Engineering Task Force RFC 1157 (May 1990).

4. *DECnet Digital Network Architecture, Common Management Information Protocol (CMIP),* Version 1.0.0 (Maynard, MA: Digital Equipment Corporation, Order No. EK-DNA01-FS-001, July 1991).

5. J. Shirley, *Guide to Writing DCE Applications* (Sebastopol, CA: O'Reilly & Associates, Inc., 1992).

6. *X/Open CAE Specification, X/Open Transport Interface (XTI),* ISBN 1-872630-29-4 (Reading, U.K.: X/Open Company Ltd., January 1992).

John R. Lawson, Jr.

Automatic, Network-directed Operating System Software Upgrades: A Platform-independent Approach

The initial system load (ISL) capability of Digital's layered-product POLYCENTER Software Distribution (formerly known as RSM) version 3.0 provides OpenVMS system managers with a network-directed tool for performing automatic operating system software upgrades. The design of the POLYCENTER Software Distribution product integrates a number of new and varied software architectures to perform the ISL. A description of the POLYCENTER Software Distribution implementation of the ISL for the OpenVMS operating system details the steps of the ISL process. The software's modular ISL mechanism can be expanded for use on other Digital and non-Digital operating systems and hardware platforms.

The POLYCENTER Software Distribution version 3.0 product provides automatic, centrally delivered, network-directed operating system software upgrades through a process called the initial system load.[1] The term *initial system load* (ISL) has existed for a number of years in various forms and has come to describe the act of loading the operating system software onto brand-new (virgin) systems without the need for locally attached tape drives or other removable media devices. This term, more loosely applied, may also be used to describe other operations such as operating system upgrades.

The ISL technology provides many advantages over the traditional means of performing upgrades. Typically, upgrades are performed one system at a time, at each console by the system manager, who must maintain the correct set of installation media for each client system's unique set of peripherals and answer each question as the upgrade procedure prompts. In a network managed by the POLYCENTER Software Distribution product, operating system upgrades are performed simultaneously. Any number of ISL operations can be invoked by using a single installation medium and often by issuing a single command. In addition, the ISL mechanism can be used for system disk maintenance operations, such as upgrade, replacement, replication, backup, or compression.

The POLYCENTER Software Distribution product can be extended for use with non-Digital operating systems and hardware platforms all controlled from its one user interface. In most cases, no backups need be performed on the clients' system disks. A halted client system, of course, must be launched into the first step manually.

This paper describes the POLYCENTER Software Distribution version 3.0 product. It begins by discussing the software environment and the software technologies used by the ISL process. It then states the project team's goals for the product. The paper next relates the ISL scheme implemented by the POLYCENTER Software Distribution version 3.0 product for the OpenVMS operating system. The paper concludes with a discussion of the details of expanding the ISL to other platforms. It is assumed that candidate operating systems are capable of at least simple task-to-task communication through the DECnet network (or some emulation), but other communication mechanisms could be devised instead.

Software Environment

The POLYCENTER Software Distribution product defines operating system software as everything on the volume that is typically called the system disk. This includes boot files, data files, configuration

files, utilities, compilers, layered products, and customizations. It even includes user directories, if they exist on that system disk.

The POLYCENTER Software Distribution product allows the individual operating system to determine the definition of upgrading the operating system software. For the OpenVMS operating system, upgrading could mean the complete replacement of the contents of the system disk volume, including the utilities, compilers, database, and customizations. For the OpenVMS AXP operating system, it could also mean the installation, without touching the rest of the volume, of only newly acquired OpenVMS operating system files and images. For other platforms, it could mean any one of several other techniques. Each platform can define what is needed to perform operating system upgrades or installations.

A network managed by the POLYCENTER Software Distribution product consists of one or more centrally located server systems; each server is responsible for performing certain operating system maintenance functions on its assigned set of client systems. The server system runs the OpenVMS operating system. Client systems run any of a number of operating systems. The POLYCENTER Software Distribution version 3.0 software supports backups, user authorizations, and layered-product installations for clients running the VAX VMS, OpenVMS VAX, OpenVMS AXP, and ULTRIX operating systems. The software includes support for ISL procedures to clients running the VAX VMS, OpenVMS VAX, and OpenVMS AXP operating systems. The software's ISL architecture, however, supports expansion to other operating systems and platforms.

Design of the POLYCENTER Software Distribution Product

The design of the POLYCENTER Software Distribution version 3.0 product integrates a number of new and varied software architectures. The software development required the cooperation and synchronization of two layered-product and two operating system development groups.

Software Technologies

No single software technology is capable of automatically upgrading system disks. Several must be used in combination. A brief description of a number of such technologies that could be used to implement the ISL process follows.

- Maintenance Operations Protocol (MOP) is a network protocol used to download system software into the memory of adjacent network nodes.
- Remote triggering enables one client system to cause another client system to reboot. The client system must have triggering enabled and have a triggering password defined and known to the server node.
- The load assist agent is a shareable image running in the context of the maintenance operations monitor (MOM) process on the server system. This code permits a server to control and customize (if necessary) the system software being downloaded to the client.
- The local area disk (LAD) protocol allows locally attached disks or container files on server systems to be presented to the local area network (LAN) for use as virtual disks on client systems. The InfoServer is the most common server of the LAD protocol. OpenVMS systems running the POLYCENTER Software Distribution product can also act as LAD servers. A system can be a LAD virtual disk.
- The processor-specific primary bootstrap is a low-level program that is loaded into the memory of a booting client. This program can be loaded from a disk drive, a tape drive, the network interconnect (NI), or read-only memory (ROM). A small but self-contained program, it is capable of communicating with the machine's console subsystem, most of the machine's internal resources, and the system disk from which it loads a secondary bootstrap or the full operating system.

  Note that some operating systems (OpenVMS included) claim that their bootstrap programs are processor-independent. However, if the operating system is under development, support will be added eventually to this bootstrap for new CPU models and/or hardware variations. Thus the processor-independent bootstrap program from an earlier version of an operating system may not support all the processor types supported by a later version of this same program. Therefore, processor independence is tied to the set of processors supported by that particular version of the operating system. For this reason, the POLYCENTER Software Distribution product specifically stores an image of the bootstrap program in a private directory alongside

the container file that houses a virtual system disk (a bootable snapshot of the version of the operating system).

- The system start-up command procedure is a command script that is responsible for bringing the recently booted operating system to its full configuration. In an ISL, this command procedure is tailored to configure only the resources needed to perform the ISL. Sometimes, limited command-level access is allowed, but seldom is full user access or timesharing permitted.
- In the OpenVMS operating system, the BACKUP/IMAGE command can duplicate a system disk either directly from disk to disk, or indirectly from a saveset file (which might be located on tape or across the network) to disk.
- Standalone BACKUP is a self-contained, diskless operating system capable of executing BACKUP/IMAGE commands but not capable of network operations.
- The SYS$UPDATE:VMSKITBLD.COM procedure in the OpenVMS operating system is used to create a generic system disk, using the current system disk software as a model.
- The POLYCENTER Software Installation (PCSI) utility is capable of creating or upgrading a system disk from a configuration file and a description file (possibly located at a remote network location) or the current system disk.

All these software technologies are utilized in the POLYCENTER Software Distribution ISL, except standalone BACKUP and SYS$UPDATE:VMSKITBLD.COM, because the former cannot be used remotely, and the latter produces uninteresting system disks. Anyone expanding the ISL, however, can use whatever techniques they choose, including those two.

Goals for the ISL Process

The development team had the following goals for the POLYCENTER Software Distribution implementation of the ISL process.

- The process must be totally automatic; only halted client systems are permitted to require human intervention.
- Multiple ISL processes must run concurrently. No specific limits should be placed on the number running in parallel (except for practical performance reasons).
- The software library must store several operating system images. They can be images of different operating systems and/or different versions of the same operating systems.
- Client systems must not be restricted to specific peripheral hardware.
- The software must make no assumptions about the hardware to which it is delivering software. Whatever configurations are legal to a particular operating system must also be supported by the POLYCENTER Software Distribution product.
- The client software must make no assumptions about the server system directing the ISL. Therefore, it would be inappropriate to store operating system images in BACKUP saveset files, which are unique to the OpenVMS operating system.
- Client software, including temporary system disks, must be taken from the clients themselves. Prepackaged operating system software is discouraged because it becomes obsolete as new versions of the operating system are developed, and because it is rarely capable of being customized by the user.
- The ISL process should be expandable to other operating systems and hardware platforms without changes to the current product.
- The POLYCENTER Software Distribution product should be able to use Digital-supplied distribution media as operating system images, such as the PCSI-based OpenVMS AXP version 6.1 CD-ROM.
- The operating system image should occupy as little disk space as possible.
- The ISL process should work over all valid DECnet network configurations. This requirement was only partially achieved: the LAD protocol works over the LAN only.

From these requirements, two achievements were gained: the totally modular organization and the compatibility with the OpenVMS AXP operating system distribution CD-ROM. The former permits support for other operating system and hardware platforms to be added incrementally. The latter enables system managers to simply load the latest copy of the distribution media, invoke the PCSI utility to record their configuration choices, and then enter a single command to upgrade all their client systems at once.

Description of the ISL Steps

Regardless of the operating system or hardware platform, the ISL process requires the following simple steps:

- Load a processor-specific primary bootstrap into the memory of the client system.
- Boot a (usually read-only) version of the operating system from some form of temporary system disk.
- Determine the parameters of the ISL to be performed.
- Move the operating system software to the target system disk.
- Initiate the cleanup, configuration, tuning, and reboot of the target system disk (which would then contain the new version of the operating system software).

Figure 1 shows the steps of the ISL process in the POLYCENTER Software Distribution Installation.

This section provides a description of each step in the ISL process, contrasting the OpenVMS implementation with the POLYCENTER Software Distribution implementation of the ISL for the OpenVMS operating system. The discussion includes both the traditional standalone installation based on the BACKUP command and the OpenVMS-defined ISL or upgrade based on the PCSI utility. The PCSI-based upgrade very closely resembles the ISL process of the POLYCENTER Software Distribution version 3.0 product.

The modular layout of the POLYCENTER Software Distribution implementation and the extension of the ISL to other operating systems are discussed in the section Platform Independence.

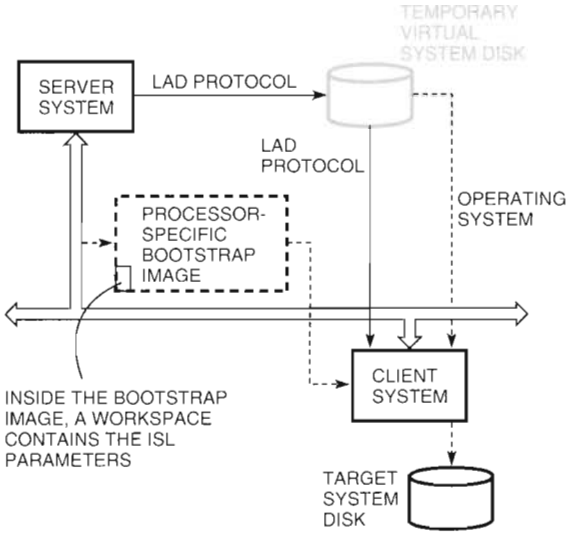


Figure 1 ISL Process in the POLYCENTER Software Distribution Installation

Processor-specific Primary Bootstrap

The primary bootstrap is responsible for establishing the connection needed between the client system and the temporary (or virtual) system disk, wherever that might be maintained.

OpenVMS Implementation If the distribution medium is local, then the ROM bootstrap or a bootstrap file on the medium is sufficient to boot the operating system contained there. If the distribution medium is served to the LAN by an InfoServer system, then the bootstrap image must be downloaded from an adjacent DECnet node with service enabled on its NI circuit common to that client, using MOP. In OpenVMS AXP, this image is called APB.SYS; in OpenVMS VAX, it is ISL_SVAX.SYS or ISL_LVAX.SYS (for small or large VAX systems, respectively).

The system manager requests the image to be downloaded (at the console of each client system) by entering special processor-dependent boot commands. The MOM process on the adjacent node drives the MOP delivery of the bootstrap image to each client. Before the connection between the client system and the temporary system disk can be established, the system manager must navigate a series of menus to select the name of the InfoServer service under which that distribution medium is presented.

POLYCENTER Software Distribution Implementation The client system boots from its NI adapter, generating a MOP load request. The server keeps the client's hardware NI address in its database so it can detect and process this request. This activates the load assist agent (LAA) under the MOM process. The LAA retrieves the various answers to the operating system configuration questions from the POLYCENTER Software Distribution library. It then passes those answers plus the operating system version-specific primary bootstrap (APB.EXE for OpenVMS AXP or ESS$ISL_VMSLOAD.EXE for OpenVMS VAX) back to the MOM process to be downloaded to the client's memory. Among these configuration answers are

the name and password of a LAD service, presented by the server in a file containing the temporary system disk. This primary bootstrap establishes the logical connection between the client system and this virtual system disk.

Temporary System Disk

The temporary system disk is a tailored copy of the operating system to be installed. This system disk is usually customized in such a way that its only purpose is to perform the ISL.

OpenVMS Implementation If it is mounted locally, the temporary system disk is the distribution medium. If the medium is presented to the client by an InfoServer service, then the temporary system disk is a virtual disk bound to that LAD service. In either case, the temporary system disk is mounted as read-only.

The operating system booted from that medium is either standalone BACKUP (for traditional installations) or OpenVMS (for PCSI-based ISL installations or upgrades). Since standalone BACKUP can perform only BACKUP operations, there is an extra, time-consuming step. The system manager must enter an appropriate BACKUP/IMAGE command to move a portion of the operating system software (the so-called REQUIRED saveset file) to the target system disk and then boot onto the target system disk (containing this partial OpenVMS operating system) to continue the installation.

POLYCENTER Software Distribution Implementation The temporary system disk always contains the full operating system to be installed. In most cases, this temporary system disk is actually a fully functional image of a model system disk taken from another client system by an earlier FETCH OPERATING_SYSTEM command.[2] The fetch process (discussed later) has replaced this temporary system disk's system start-up command procedure with a script that runs the remaining steps of the ISL process.

Previous versions of this product (also known as RSM) included a prepackaged temporary system disk with a fixed contents that was built by hand. Software developers routinely captured the latest versions of OpenVMS system disks inside boot container files as small as 14,000 blocks! Although an interesting academic achievement, this proved to be an impractical approach. Digital releases new processors from time to time, and each new processor requires a new minimum version of the OpenVMS operating system. The system disks captured in the boot container could not be easily upgraded in the field. An engineering change order was required for the POLYCENTER Software Distribution product each time support was added to the OpenVMS system for a new processor.

These previous versions also stored the operating system image in a BACKUP saveset file. This method could be more space-efficient (page, swap, and dump files consume no space in a saveset file), but it violates one of the design goals.

In version 3.0, the software developers eliminated the concept of separate BACKUP saveset files and boot containers. Since the operating system support for the new processors exists in the software saved in the operating system image, the clients can be booted directly from that image. The POLYCENTER Software Distribution version 3.0 product stores the image of the model system disk directly into a container file. This approach produced an interesting side effect. If a particular processor is not supported by the version of OpenVMS saved in the operating system image, it is not possible to boot that processor into the ISL. As a result, an older version of the OpenVMS operating system cannot be installed on hardware that requires a newer version.

Parameters of the ISL

When operating system software is being installed, system configuration choices must be selected from a number of variables. At a minimum, the name of the target system disk must be known. Answers might also be needed for questions such as: which subsets of the operating system files are to be installed? The ISL procedure must be capable of obtaining these answers, either by prompting a user at the console of the client system or by some automatic means.

OpenVMS Implementation At this point, the OpenVMS operating system is running, and a special system start-up command procedure has control. The system manager now answers a series of prompts at the console. Only rarely does the upgrade procedure ask all its questions at once (and state that it is finished asking questions) before commencing any time-consuming tasks. If it did, the system manager could leave the console of one machine to move to the console of the next machine and so on. In this way, multiple upgrades could be performed concurrently.

POLYCENTER Software Distribution Implementation The parameters of the ISL were downloaded along with the primary bootstrap image. The system start-up procedure of the ISL executes a program that locates the list of parameters in memory and returns them as logical names, which are easier for command procedures to manipulate. (Other operating systems would use their own easily accessible data storage mechanisms.)

The system start-up procedure starts the DECnet networking software and establishes a network connection with the POLYCENTER Software Distribution server system, permitting access to larger amounts of data than might fit into the bootstrap image. The BACKUP saveset file used by previous versions of the POLYCENTER Software Distribution product was accessed through this DECnet connection.

Move the Operating System Software

Each operating system has specific requirements for creating or duplicating system disks. This step uses the client operating system's standard procedure to duplicate or upgrade the target system disk, generally using the temporary system disk as its source (or model). However, another means, such as network files or library files, may be used.

OpenVMS Implementation From this point, there is no difference between an upgrade and an installation using the traditional standalone OpenVMS mechanisms.

The OpenVMS mechanism now performs a series of complex file replacements in a peculiar order, which requires several reboots to complete. This maximizes the existing free space on the target system disk. After all the reboots have completed, the old operating system files will have been deleted, and the new files will have been delivered.

The PCSI-based upgrade does not need to perform the several reboots, since the target system disk is treated as a data disk. Its operating system files are simply replaced with new versions taken from the temporary system disk. This is one reason that the PCSI-based OpenVMS upgrade is faster than the traditional OpenVMS upgrade.

POLYCENTER Software Distribution Implementation Since full OpenVMS (including DECnet) is running, all the resources of the OpenVMS operating system are available for manipulating the target system disk, which is also treated as a data disk. Alternatives such as VMSKITBLD.COM (which creates duplicate basic system disks), the BACKUP/IMAGE command (which duplicates system disks in their entirety), and the PCSI utility (which upgrades system disks in place) could be utilized at this point.

The BACKUP/IMAGE command moves the image of the temporary system disk to the target system disk. The PCSI utility replaces the operating system files on the target system disk with the new operating system files from the temporary system disk. In the BACKUP/IMAGE case, any system-specific customizations or layered-product files that were saved into the container file by the FETCH OPERATING_SYSTEM process are now in place. In the PCSI case, however, all system-specific customizations or layered-product files are left undisturbed.

Cleanup, Configuration, Tuning, and Reboot

Any final changes needed before allowing the client to use its new system disk are performed during the cleanup, configuration, tuning, and reboot phase. The client now boots from its newly upgraded target system disk, and the temporary system disk is no longer needed.

OpenVMS Implementation As a final step, the AUTOGEN procedure tunes the operating system parameters to the hardware on which it is intended to run. Any other configuration issues (such as the network node name and address) remain as exercises for the system manager to perform at some later time. The system reboots one last time. For traditional installations, this reboot may have been the fifth or sixth. Some of these may have been manual reboots, which require the system manager to issue nonstandard, processor-specific console commands.

POLYCENTER Software Distribution Implementation When the BACKUP/IMAGE command is used, customizations specific for the ISL, which are all stored under the [RSM0.] directory tree, must be removed from the target system disk, which, before this step, is a perfect image of the temporary system disk. In addition, the DECnet software must be reconfigured. The DECnet databases still contain the configurations saved in the temporary system disk; these must be updated to reflect the hardware on this client. As a final step, the client reboots onto the target system disk, and the temporary system disk is no longer required. With the PCSI-based ISL, no cleanup is required.

Fetching and Installing Operating System Software

The POLYCENTER Software Distribution product keeps images of model operating systems in its private software library. The act of placing software into the library is called a fetch. The act of delivering that software to a client system is called an install. The operating system commands are FETCH OPERATING_SYSTEM and INSTALL or UPGRADE OPERATING_SYSTEM.[3] A model system disk cannot be installed without first being fetched from a client system or suitable distribution medium.

The Fetch Operation

The FETCH OPERATING_SYSTEM command takes a parameter that is the symbolic name of the operating system to be fetched. The POLYCENTER Software Distribution product uses and records this symbolic name because it is the key to a naming scheme used to activate program modules later.

Table 1 lists several symbolic names and the operating systems they might represent. It is important to remember that there is no built-in mapping between these names and the operating systems to which they are mapped. This list is a theoretical sampling of what mappings could be configured on a particular server system.

When processing an INSTALL OPERATING_SYSTEM AVMS command, the POLYCENTER Software Distribution product uses the OpenVMS system run-time library routine LIB$FIND_IMAGE_SYMBOL in order to dynamically activate the shareable image SYS$SHARE:RSM$ISL_INSTALL-AVMS.EXE. This image is called the ISL Director. It is used for both fetch and install operations. The POLYCENTER Software Distribution product calls the ISL Director routine RSM$ISL_FETCH and passes to it a context data structure (described in the section Platform Independence). This routine uses the software's remote command execution agent (CEA) to issue Digital command language (DCL) commands on the client system. Non-OpenVMS clients would need to implement their own communications mechanism, so that the server system could direct the client to perform any required actions.

These DCL commands cause the client to mount the LAD virtual disk presented from the fetch toolkit container file RSM$SDS_DATA:RSM$FETCH-AVMS.DSK. The client executes the command procedure [RSMV3.0]RSM$ISL_BOOT-AVMS.COM from the fetch toolkit virtual disk. This command procedure

- Determines the size of the client's system disk
- Reports that system disk size to the ISL Director

Table 1 Required Files for Sample Operating Systems

Symbolic Name	Operating System	Required ISL Files
VMS	OpenVMS VAX or VAX VMS	SYS$SHARE:RSM$ISL_INSTALL-VMS.EXE RSM$SDS_DATA:RSM$FETCH-VMS.DSK SYS$SHARE:RSM$ISL_LAA-VMS.EXE
AVMS	OpenVMS AXP	SYS$SHARE:RSM$ISL_INSTALL-AVMS.EXE RSM$SDS_DATA:RSM$FETCH-AVMS.DSK SYS$SHARE:RSM$ISL_LAA-AVMS.EXE
ULTRIX	ULTRIX	SYS$SHARE:RSM$ISL_INSTALL-ULTRIX.EXE RSM$SDS_DATA:RSM$FETCH-ULTRIX.DSK SYS$SHARE:RSM$ISL_LAA-ULTRIX.EXE
OSF1	OSF/1	SYS$SHARE:RSM$ISL_INSTALL-OSF1.EXE RSM$SDS_DATA:RSM$FETCH-OSF1.DSK SYS$SHARE:RSM$ISL_LAA-OSF1.EXE
VMS5	OpenVMS VAX with DECnet Phase V	SYS$SHARE:RSM$ISL_INSTALL-VMS5.EXE RSM$SDS_DATA:RSM$FETCH-VMS5.DSK SYS$SHARE:RSM$ISL_LAA-VMS5.EXE
AVMS5	OpenVMS AXP with DECnet Phase V	SYS$SHARE:RSM$ISL_INSTALL-AVMS5.EXE RSM$SDS_DATA:RSM$FETCH-AVMS5.DSK SYS$SHARE:RSM$ISL_LAA-AVMS5.EXE
WINDOWS	MS-DOS running Microsoft Windows	SYS$SHARE:RSM$ISL_INSTALL-WINDOWS.EXE RSM$SDS_DATA:RSM$FETCH-WINDOWS.DSK SYS$SHARE:RSM$ISL_LAA-WINDOWS.EXE

The server creates an appropriately sized LAD container file to receive the snapshot of the client's system disk and serves it to the client.

- Mounts the new virtual disk
- Issues a BACKUP/IMAGE command to copy the system disk to the virtual disk
- Provides the server with access to the processor-specific primary bootstrap image (APB.EXE)

  The server saves the APB.EXE image in its library alongside the newly created container file.
- Customizes the virtual disk so it can be used as the temporary system disk during an ISL

  The boot command procedure uses programs and command procedures from the fetch toolkit virtual disk to accomplish this step. In a FETCH OPERATING_SYSTEM AVMS, this final step includes creating a special system root [RSM0.SYSEXE], placing a private system start-up command procedure [RSMV3.0]RSM$ISL_STARTUP-AVMS.COM, installing a program to retrieve the parameters of the ISL [RSMV3.0]RSM$ISL_CLIENT-AVMS.EXE, and installing a command procedure to remove these customizations [RSMV3.0]RSM$ISL_CLEANUP-AVMS.COM.

The two virtual disks are then dismounted, and the server closes the container file and makes it available, write-protected, for ISL operations. These LAD services can be accessed with binary passwords known only to POLYCENTER Software Distribution servers, so no casual access to the data contained within is ever allowed.

The Install Operation

The POLYCENTER Software Distribution product retrieves the symbolic name of the operating system (e.g., AVMS) from the database. The software product uses the symbolic name to activate the ISL Director image (SYS$SHARE:RSM$ISL_INSTALL-AVMS.EXE) and passes control to its universal routine RSM$ISL_INSTALL. This routine enables the LAA (SYS$SHARE:RSM$ISL_LAA-AVMS.EXE) and prepares a data file RSM$SDS_WORK:ISL_client.DAT for use by the LAA after the client system requests it to be downloaded.

If a DECnet connection is possible between the server system and the client system, then the command execution agent issues appropriate shutdown and reboot commands to launch the ISL. If not, the POLYCENTER Software Distribution process assumes that the client is halted and that the system manager will launch the client into the ISL manually.

When the MOM process detects the client's NI address, it activates the LAA and passes control to the routine at offset 0000 in the image. The parameters to this procedure call (which are described in the section Platform Independence) include the node name of the client system and the address of a callback routine used to deliver the bytes of the bootstrap image to the client. The callback routine

- Reads the RSM$SDS_WORK:ISL_client.DAT file (described in the section Platform Independence)
- Retrieves the processor-specific bootstrap image (APB.EXE) from the library
- Locates and writes the parameters of the ISL into the bootstrap image's work space
- Releases these bytes to MOM for delivery to the client

Once this is downloaded, the server system assumes a passive role, waiting for the client to announce its own completion.

The processor-specific bootstrap image has control of the client system. It locates the LAD service name and password in the parameters of the ISL to establish the connection to the temporary virtual system disk (which is being presented by the server system) and boots the OpenVMS AXP operating system.

The system start-up command procedure (RSM$ISL_STARTUP-AVMS.COM) then receives control and

- Starts enough of the OpenVMS operating system to mount local disks and start the DECnet networking software
- Executes the program RSM$ISL_CLIENT-AVMS.EXE to retrieve the ISL parameters

With the parameters of the ISL stored in logical names, the system start-up procedure then

- Configures the target system disk
- Initializes the target system disk if necessary
- Starts the DECnet networking software
- Solicits further instructions (if any) from the server system
- Issues a BACKUP/IMAGE command to move the operating system software from the temporary system disk to the target system disk

- Executes the RSM$ISL_CLEANUP-AVMS.COM command procedure to remove the customizations specific for the ISL

The target system disk now appears to be identical to the model system disk just before the fetch operation.

The UPGRADE OPERATING_SYSTEM and FETCH CONFIGURATION Commands

The contents of the PCSI-installable distribution medium for OpenVMS AXP bears a striking resemblance to a POLYCENTER Software Distribution temporary system disk. This is no coincidence. The OpenVMS AXP development team modeled the distribution medium after the POLYCENTER Software Distribution boot container, so the product would be plug-compatible. The obvious difference, however, is that the system start-up procedure invokes the PCSI utility instead of the BACKUP/IMAGE command.

The client system boots from the distribution medium under the direction of the POLYCENTER Software Distribution product. Next the procedure starts the DECnet network software using the parameters of the ISL. Then the PCSI configuration answers are taken from the server system rather than being prompted manually at the console. Everything else is the same.

Before any of this is possible, however, the system manager invokes the PCSI utility to record the answers to all the configuration questions using the RSM$TRIAL_INSTALL.COM command procedure. The PCSI configuration file is then inserted in the POLYCENTER Software Distribution library using the FETCH CONFIGURATION command.

Note that when recording configuration files, the PCSI utility permits users to defer answers until installation time. Unfortunately, because of the product's stipulation that no human intervention be required, such deferrals cause the ISL to fail.

Platform Independence

The following section details how the ISL process can be expanded to other platforms and operating systems. Table 1 gives a sample list of symbolic names and their corresponding operating systems. The POLYCENTER Software Distribution version 3.0 kit provides only the VMS (for the VAX VMS and the OpenVMS VAX operating systems) and the AVMS (for the OpenVMS AXP operating system) ISL kits.

To add ISL support for other operating systems and/or hardware platforms, the following requirements must be met.

- The operating system must be bootable from a read-only LAD virtual disk. (Among others, the MS-DOS, ULTRIX, and DEC OSF/1 operating systems are known to have this capability.)
- The hardware platform must be MOP downloadable. (Most Digital processors have this capability.)
- The operating system's processor-specific bootstrap image must have an LAA-writeable scratch area for the parameters of the ISL.
- The parameters of the ISL must be retrievable by the operating system's system start-up command procedure.
- The operating system must have a mechanism for moving the contents of the temporary system disk to the target system disk, which will never be identical media. (Most operating systems have this capability.)
- The ISL Director shareable image (SYS$SHARE:RSM$ISL_INSTALL-*opera*.EXE), containing entry points RSM$ISL_FETCH and RSM$ISL_INSTALL, must be active on the server system (running OpenVMS).
- The contents of the fetch toolkit container file (RSM$SDS_DATA:RSM$FETCH-*opera*.DSK) need be known only to the ISL Director. This file resides on the server system (running OpenVMS) but is read only by the client system and only during a fetch operation.
- The load assist agent (SYS$SHARE:RSM$ISL_LAA-*opera*.EXE) must be capable of delivering the operating system's processor-specific primary bootstrap image (plus the parameters of the ISL) to the client system, which runs on the server system (running OpenVMS).

Table 1 lists the names of the files required to support various operating systems. Note the naming scheme for the files. Each set of three files, which compose a single ISL kit, implements the entire ISL fetch and install functionality. The ISL Director routine RSM$ISL_FETCH works in conjunction with the fetch toolkit. The ISL Director routine RSM$ISL_INSTALL works in conjunction with the load assist agent. Table 2 gives the naming convention used for all resources shared between these three files. The term *opera* identifies the symbolic name of the operating system. The term *server* identifies the DECnet node name of the server

Table 2 Naming Conventions Used by ISL Resources

Name	Description
RSM$ISL_INSTALL-*opera*.EXE	Shareable image containing the ISL Director routines, which runs on the POLYCENTER Software Distribution server (running OpenVMS).
RSM$ISL_LAA-*opera*.EXE	Shareable image containing the load assist agent, which runs on the POLYCENTER Software Distribution server (running OpenVMS).
RSM$FETCH-*opera*.DSK	Container file containing the fetch toolkit, which resides on the POLYCENTER Software Distribution server system (running OpenVMS) but is read only by the client system.
RSM$FETCH_*server-opera*	LAD service name for the fetch toolkit, which is served by the POLYCENTER Software Distribution server (running OpenVMS) to the client.
RSM$ISL_BOOT-*opera*.COM	Command procedure responsible for actually performing the save of the operating system software from the client's system disk to the virtual disk, which runs on the client system.
RSM$SDS_OS_LIBRARY: [opsys.OPERSYS]SYS0.DSK	Container file for the fetched operating system, which resides on the POLYCENTER Software Distribution server system (running OpenVMS). (This directory may also be used to store the bytes of the processor-specific bootstrap image so the load assist agent has easy access.)
RSM$ISL_*server-opsys*	LAD service name of the temporary system disk containing the fetched operating system image, which is served from the POLYCENTER Software Distribution server system (running OpenVMS) to the client system.
RSM$ISL_STARTUP-*opera*.COM	Command procedure responsible for actually delivering the operating system software from the temporary system disk to the target system disk. It runs on the client system but is booted from the temporary system disk.
RSM$ISL_CLEANUP-*opera*.COM	Command procedure for removing customizations specific to the initial system load from a temporary system disk. It runs on the client system but is booted from the temporary system disk.
RSM$ISL_*server*	LAD service name of the VAX "boot container" for operating systems fetched prior to version 3.0, which is served from the POLYCENTER Software Distribution server system (running OpenVMS) to the client system (which in this case must be running OpenVMS VAX or VAX VMS).
RSM$ISL_*server*_EVMS	LAD service name of the AXP "boot container" for operating systems fetched prior to version 3.0, which is served from the POLYCENTER Software Distribution server system (running OpenVMS) to the client system (which in this case must be running OpenVMS AXP).

system, and the term *opsys* identifies the user-defined pseudonym for the fetched operating system image.

The ISL Director

The ISL Director is a shareable image activated by LIB$FIND_IMAGE_SYMBOL; therefore it need not have transfer vectors, as long as the two required entry points are declared UNIVERSAL. These two routines are called in user mode. They are passed a single parameter, the address of a data structure called the QENTRY.

The pertinent fields of the QENTRY data structure passed to RSM$ISL_FETCH are

```
.
.
.
char pseudonym[64];
.
.
.
char client_node[128];
.
.
.
char library_node[128];
.
.
.
char opera_house[8];
.
.
.
```

and the pertinent fields of the QENTRY data structure passed to RSM$ISL_INSTALL are

```
.
.
.
char ethernet[19];
.
.
.
char client_node[128];
.
.
.
char library_node[128];
.
.
.
char opera_house[8];
.
.
.
```

In both routines, the field called opera_house contains the symbolic name of the operating system (e.g., AVMS).

RSM$ISL_FETCH is responsible for copying a bootable snapshot of the client's system disk into the LAD container file SYS0.DSK. The LAD virtual disk should be organized into the native format of the operating system being fetched. The server system will never attempt to read these files. To the server system, this container is simply a large series of bytes, whose meaning (to the client system) is unimportant. This routine is responsible for obtaining the size of the container file to be created, creating that container file, and then serving it, writeable, to the LAD. Once the fetch operation has concluded, the container should be served again in read-only format.

RSM$ISL_INSTALL is responsible for enabling the LAA for the new client system. Since the LAA runs under the MOM process, which is a non-POLYCENTER Software Distribution environment, this routine should also collect any and all information (such as the DECnet node name and address of the server system) needed by the LAA, and store that information in the file RSM$SDS_WORK:ISL_client.DAT. The content of this file is shared only between RSM$ISL_INSTALL and the LAA; therefore, the format of the file is implementation-dependent.

The Fetch Toolkit

The fetch toolkit is also a LAD virtual disk organized in a format that is native to the client's operating system. Again, the server system will never read this virtual volume. This virtual volume contains the native operating system pieces necessary to save a snapshot of the model system disk, make it bootable as the temporary system disk, and restore it to its original state. These are usually three separate command procedures. The command procedure that saves the system disk image must also store the bytes of the operating system's primary bootstrap image for future access by the LAA.

The Load Assist Agent

The LAA delivers the bytes of the processor-specific primary bootstrap image to the client system. The MOM process activates this shareable image dynamically, but not using LIB$FIND_IMAGE_SYMBOL. Therefore, the one required entry point to this image must occur at offset 0000 in the image. (The name of the entry point is unimportant.) This is best accomplished using a single transfer vector.

This routine is called in user mode with three parameters, the addresses of three data structures: the MOMIDB, the MOMARB, and the MOMODB.

The offset MOMIDB$A_PARAM_DSC contains any text from the NCP load assist parameter field. This field contains arbitrary text that RSM$ISL_INSTALL placed there. Normally, this field contains a handle used to retrieve the file RSM$SDS_WORK:ISL_client.DAT. A good handle is the DECnet node name of the client system.

The offset MOMARB$A_SEND_DATA is the address of a routine to deliver data to the client. The LAA need only collect and/or generate the data to be delivered to the client; this callback routine delivers it to the client. Its two parameters are a string descriptor identifying which and how many bytes are to be delivered, and the relative address in the client's memory to place these bytes. This callback routine may be called repetitively.

The offset MOMODB$L_TRANSFER_ADDRESS must be filled with the relative transfer address of the processor-specific bootstrap image that was loaded into the client's memory by MOMARB$A_SEND_DATA. For OpenVMS VAX, this offset is traditionally zero, because certain older VAX processors are not capable of using any other value. That is one reason why the transfer address for ISL_SVAX.SYS is always zero.

Summary

The ISL mechanism installs, maintains, and upgrades operating system software. These simple descriptions provide the framework for expanding the ISL process implemented in the POLYCENTER Software Distribution version 3.0 product to platforms other than OpenVMS VAX and OpenVMS AXP operating systems. This expansion can make work easier for system managers of multiple platforms and may even start a de facto standard for performing operating system upgrades.

Acknowledgments

I would like to thank Richard Bishop and Charlie Hammond in the OpenVMS AXP Development Group for allowing me to unify the POLYCENTER Software Distribution version 3.0 ISL and the PCSI-based OpenVMS AXP version 6.1 upgrade.

Note and References

1. POLYCENTER Software Distribution is the new name for Digital's Remote System Manager product. The installed software continues to use its traditional acronym RSM.

2. *POLYCENTER Software Distribution Management Guide* (Maynard, MA: Digital Equipment Corporation, Order No. AA-JG05E-TE, May 1994).

3. *POLYCENTER Software Distribution Command Reference* (Maynard, MA: Digital Equipment Corporation, Order No. AA-JG03E-TE, May 1994).

Further Readings

The following technical papers were written by Digital authors:

R. Abugov and K. Zinke, "Wafer Level Tracking Enhances Particle Source Isolation in a Manufacturing Environment," *Fifth Annual IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop* (November 1994).

J. Card, A. McGowan, and C. Reed, "Neural Network Approach to Automated Wirebond Defect Classification," *ASME Proceedings of the Artificial Neural Networks in Engineering (ANNIE '94) Conference* (November 1994).

S. Cheung, D. Jensen, and G. Mooney, "Ultra-High Purity Gas Distribution Systems for Sub 0.5um ULSI Manufacturing," *Fifth Annual IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop* (November 1994).

R. Collica, B. Cantell, and J. Ramírez, "Statistical Analysis of Particle/Defect Data Experiment Using Poisson and Logistic Regression," *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems* (October 1994).

B. Doyle, K. Mistry, and C-L. Huang, "Analysis of Gate Oxide Thickness Hot Carrier Effects in Surface Channel P-MOSFET's," *IEEE Transactions on Electron Devices* (January 1995).

J. Edmondson, "Internal Organization of the Alpha 21164," *IEEE First International Symposium on High-performance Computer Architecture (HPCA)* (January 1995).

L. Elliott, D. Paine, and J. Rose, "The Microstructure and Electromigration Behaviour of Al-0.35%Pd Interconnects," *Materials Research Society Symposium Proceedings: Materials Reliability in Microelectronics IV Symposium* (April 1994).

C. Gordon and K. Roselle, "An Efficient and Accurate Method for Estimating Crosstalk in Multiconductor Coupled Transmission Lines," *IEEE Third Topical Meeting on the Electrical Performance of Electronic Packaging* (November 1994).

D. Heimann, "Using Complexity-Tracking in the Software Development Process," *Thirty-second Annual Spring Reliability Symposium* (April 1994).

A. John, "Dynamic Vnodes: Design and Implementation," *USENIX 1995 Technical Conference on UNIX and Advanced Computing Systems* (January 1995).

D. Jones and V. Murthy, "Advancing Reliability with State of the Art Software Tools," *University of Manchester School of Engineering Third Reliability Software Seminar and Workshop* (December 1994).

N. Khalil, J. Faricelli, and D. Bell, "The Extraction of Two-Dimensional MOS Transistor Doping via Inverse Modeling," *IEEE Electron Device Letters* (January 1995).

A. Labun, "Profile Simulation of Electron Cyclotron Resonance Planarization of an Interlevel Dielectric," *Journal of Vacuum Science and Technology B (JVST B)* (November/December 1994).

K. Mistry and B. Doyle, "How Do Hot Carriers Degrade N-Channel MOSFETs?," *IEEE Circuits and Devices* (January 1995).

C. Ozveren, R. Simcoe, and G. Varghese, "Reliable and Efficient Hop-by-Hop Flow Control," *ACM SIGCOMM 94* (October 1994).

R. Razdan and M. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units," *Proceedings of the Twenty-seventh Annual International Symposium on Microarchitecture (MICRO-27)* (December 1994).

R. Rios and N. Arora, "Determination of Ultra-Thin Gate Oxide Thickness for CMOS Structures Using Quantum Effects," *IEEE International Electron Devices Meeting/IEDM Technical Digest* (December 1994).

N. Sullivan, "Semiconductor Pattern Overlay," *Proceedings of the International Society of Photo-Optical Instrumentation Engineers (SPIE) Microelectronic Processing: Integrated Circuit Metrology and Process Control (Critical Review)* (September 1994).

B. Thomas, "OpenVMS I/O Concepts: CSR Access," *Digital Systems Journal* (November/December 1994).

Recent Digital U.S. Patents

The following patents were recently issued to Digital Equipment Corporation. Titles and names supplied to us by the U.S. Patent and Trademark Office are reproduced exactly as they appear on the original published patent.

5,208,692	D. McMahon	High Bandwidth Network Based on Wavelength Division Multiplexing
5,208,768	E. Simoudis	Expert System Including Arrangement for Acquiring Redesign Knowledge
5,210,854	A. Beaverson and T. Hunt	System for Updating Program Stored in EEPROM by Storing New Version into New Location and Updating Second Transfer Vector to Contain Starting Address of New Version
5,210,865	S. Davis, W. Goleman, and D. Thiel	Transferring Data between Storage Media While Maintaining Host Processor Access for I/O Operations
5,217,198	V. Samarov, W. Pauplis, and G. Doumani	Uniform Spatial Action Shock Mount
5,218,678	B. Kelleher and S-S. Chow	System and Method for Atomic Access to an Input/Output Device with Direct Memory Access
5,220,661	J. Wray, A. Mason, P. Karger, P. Robinson, W-M. Hu, and C. Kahn	System and Method for Reducing Timing Channels in Digital Data Processing Systems
5,224,206	E. Simoudis	System and Method for Retrieving Justifiably Relevant Cases from a Case Library
5,224,884	M. Singer, R. Noffke, and D. Gilmour	High Current, Low Voltage Drop Separable Connector
5,233,684	R. Ulichney	Method and Apparatus for Mapping a Digital Color Image from a First Color Space to a Second Color Space
5,235,697	S. Steely and J. Zurawski	Set Prediction Cache Memory System Using Bits of the Main Memory Address
5,239,634	B. Buch and C. MacGregor	Memory Controller for Engineering/Dequeuing Process
5,239,637	S. Davis, W. Goleman, and D. Thiel	Digital Data Management System for Maintaining Consistency of Data in a Shadow Set
5,241,564	J. Tang and J.L. Yang	Low Noise, High Performance Data Bus System and Method
5,242,761	Y. Uchiyama	Magnetic Recording Medium and Method of Manufacture Thereof
5,243,241	C-H. Wang	Totally Magnetic Fine Tracking Miniature Galvanometer Actuator
5,247,464	R. Curtis	Node Location by Differential Time Measurements
5,247,618	S. Davis, W. Goleman, D. Thiel, R. Bean, and J. Zahrobsky	Transferring Data in a Digital Data Processing System
5,251,147	J. Finnerty	Minimizing the Interconnection Cost of Electronically Linked Objects
5,251,227	T. Bissett, W. Bruckert, J. Munzer, D. Kovalcin, and M. Norcross	Resets for a Fault Tolerant, Dual Zone Computer System
5,253,249	J. Fitzgerald and D. Shuda	Bidirectional Transceiver for High Speed Data System
5,253,353	J.C. Mogul	System and Method for Efficiently Supporting Access to I/O Devices through Large Direct-mapped Data Caches
5,257,264	H. Yang and K.K. Ramakrishnan	Automatically Deactivated No-owner Frame Removal Mechanism for Token Ring Networks
5,261,077	J.R. Duval, K.R. Peterson, and T.E. Hunt	Configurable Data Path Arrangement for Resolving Data Type Incompatibility (This case is related to PD89-0300.)

5,261,085	L.B. Lamport	Fault-tolerant System and Method for Implementing a Distributed State Machine
5,265,092	S.R. Soloway, A.G. Lauck, and G. Verghese	Synchronization Mechanism for Link State Packet Routing
5,265,257	R.J. Simcoe and R.E. Thomas	Fast Arbiter Having Easy Scaling for Large Numbers of Requesters, Large Numbers of Resource Types with Multiple Instances of Each Type and Selectable Queueing Disciplines
5,274,811	A. Borg and D.W. Wall	Method for Quickly Acquiring and Using Very Long Traces of Mixed System and User Memory References
5,276,712	J.D. Pearson	Method and Apparatus for Clock Recovery in Digital Communication Systems
5,276,809	J.K. Grooms, R.L. Sites, L.A. Chisvin, and D.W. Smelser	Method and Apparatus for Capturing Real-time Data Bus Cycles in a Data Processing System
5,276,828	J. Dion	Methods of Maintaining Cache Coherence and Processor Synchronization in a Multiprocessor System Using Send and Receive Instructions
5,276,851	C. Thacker and D. Conroy	Automatic Writeback and Storage Limit in a High-performance Frame Buffer and Cache Memory System
5,276,874	R.G. Thomson	Method for Creating a Directory Tree in Main Memory Using an Index File in Secondary Memory
5,278,974	R. Ramanujan, P.J. Lemmon, and J.C. Stickney	Method and Apparatus for the Dynamic Adjustment of Data Transfer Timing to Equalize the Bandwidths of Two Buses in a Computer System Having Different Bandwidths
5,280,478	H. Yang, P.W. Ciarfella, K.K. Ramakrishnan	No-owner Frame and Multiple Token Removal Mechanism for Token Ring Networks
5,280,575	C.A. Young and N.F. Jacobson	Apparatus for Cell Format Control in a Spreadsheet
5,280,582	H. Yang, K.K. Ramakrishnan, and A. Lauck	No-owner Frame and Multiple Token Removal for Token Ring Networks
5,280,627	J.E. Flaherty and A. Abrahams	Remote Bootstrapping a Node over Communication Link by Initially Requesting Remote Storage Access Program Which Emulates Local Disk to Load Other Programs
5,283,857	E. Simoudis	Expert System Including Arrangement for Acquiring Redesign Knowledge
5,283,873	S.C. Steely and D.J. Sager	Next Line Prediction Apparatus for a Pipelined Computer System
5,287,438	B.M. Kelleher	System and Method for Drawing Antialiased Polygons
5,287,485	L. Umina and R. Anselmo	Digital Processing System Including Plural Memory Devices and Data Transfer Circuitry
5,287,534	T. Reuther	Correcting Crossover Distortion Produced When Analog Signal Thresholds Are Used to Remove Noise from Signal
5,291,494	T. Bissett, W. Bruckert, and J. Melvin	Method of Handling Errors in Software
5,293,620	W. Barabash and W.S. Yerazunis	Method and Apparatus for Scheduling Tasks in Repeated Iterations in a Digital Data Processing System Having Multiple Processors
5,296,392	G.J. Grula and W.C. Metz	Method for Forming Trench Isolated Regions with Sidewall Doping
5,297,269	D. Donaldson, M. Howard, D. Orbits, J. Parchem, D. Robinson, and D. Williams	Cache Coherency Protocol for Multi Processor Computer System

5,298,464	R.W. Doe, R.D. Gates, D.P. Goddard, S.C. Hsu, and R.L. Schlesinger	Method of Manufacturing Tape Automated Bonding Semiconductor Package
5,301,327	W. McKeeman and S. Aki	Virtual Memory Management for Source-code Development System
5,303,382	B. Buch and C. MacGregor	Arbiter with Programmable Dynamic Request Prioritization
5,303,391	R.J. Simcoe and R.E. Thomas	Fast Arbiter Having Easy Scaling for Large Numbers of Requesters, Large Numbers of Resource Types with Multiple Instances of Each Type and Selectable Queueing Disciplines
5,313,387	W.M. McKeeman and S. Aki	Re-execution of Edit-compile-run Cycles for Changed Lines of Source Code, with Storage of Associated Data in Buffers
5,313,464	F.H. Reiff	Fault Tolerant Memory Using Bus Bit Aligned Reed-Solomon Error Correction Code Symbols
5,313,641	R.J. Simcoe and R.E. Thomas	Fast Arbiter Having Easy Scaling for Large Numbers of Requesters, Large Numbers of Resource Types with Multiple Instances of Each Type and Selectable Queueing Disciplines
5,315,480	V. Samarov, G. Doumani, and R. Larson	Conformal Heat Sink for Electronic Module
5,317,708	R. Edgar	Content Addressable Memory
5,319,651	R. Helliwell, R. Lary, B. Edem, and J. Johnston	Data Integrity Features for a Sort Accelerator
5,321,724	B. Long and M.J. Hynes	Interference Suppression System
5,321,841	M.C. Ozur, S.M. Jenness, J.W. Kelly, J.J. Walker, and J.A. East	Server Impersonation of Client Processes in an Object-based Computer Operating System
5,325,531	W.M. McKeeman and S. Aki	Incremental Compiler for Source Code Development System
5,327,368	R.A. Eustace and J.S. Leonard	Chunky Binary Multiplier and Method of Operation
5,327,557	J.P. Emmond	Single-keyed Indexed File for TP Queue Repository
5,330,881	A. Sidman and S. Fung	Microlithographic Method for Producing Thick Vertically Walled Photoresist Patterns
5,337,404	P. Beaudelaire, M. Gangnet, J. Herve, T. Pudet, and J.V. Thong	Process for Making Computer-aided Drawings
5,339,449	P.A. Karger, A.H. Mason, J.C.R. Wray, P.T. Robinson, A.L. Priborsky, C.E. Kahn, and T.E. Leonard	System and Method for Reducing Storage Channels in Disk Systems
5,345,587	L.G. Fehskens, C. Strutt, S. Wong, J.F. Callander, P.H. Burgess, K.J. Nelson, M.J. Guertin, D.L. Smith, M.W. Sylor, K.W. Chapman, R.C. Schuchard, S.I. Goldfarb, R.R.N. Ross, L.B. O'Brien, P.J. Trasatti, D.O. Rogers, B.M. England, J.L. Lemmon, R.L. Rosenbaum, and additional inventors	Extensible Entity Management System Including a Dispatching Kernel and Modules Which Independently Interpret and Execute Commands
5,345,588	R. Peterson, B. Schreiber, and S. Greenwood	Thread Private Memory Storage for Multithread Digital Data Processing

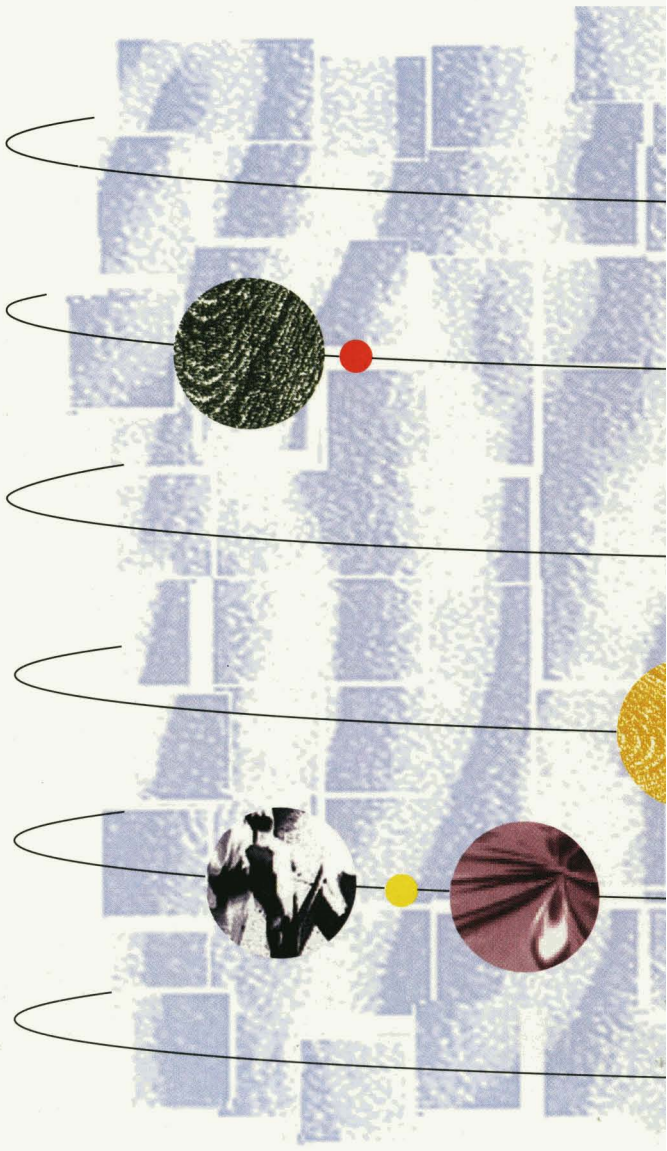
Call for Authors from Digital Press

Digital Press has become an imprint of Butterworth-Heinemann, a major international publisher of professional books and a member of the Reed Elsevier group. Digital Press remains the authorized publisher for Digital Equipment Corporation: the two companies are working in partnership to identify and publish new books under the Digital Press imprint and create opportunities for authors to publish their work.

Digital Press remains committed to publishing high-quality books on a wide variety of subjects. We would like to hear from you if you are writing or thinking about writing a book.

Contact: Frank Satlow
Publisher
Digital Press
313 Washington Street
Newton, MA 02158
Tel: (617) 928-2649
Fax: (617) 928-2640
fps@world.std.com

digital™



ISSN 0898-901X