ACMSxp Open Distributed Transaction Processing

by

Robert K. Baafi, J. Ian Carrie, William B. Drury, and Oren L. Wiesler

ABSTRACT

Digital's ACMSxp portable transaction processing (TP) monitor supports open TP standards and provides an environment for the development, execution, and administration of robust, distributed, client-server applications. The ACMSxp TP monitor supports the Structured Transaction Definition Language, a modular language that simplifies the development of transactional applications. ACMSxp software is layered on the Open Software Foundation's Distributed Computing Environment (DCE) and supports XA-compliant databases and other resource managers by using the Encina toolkit from Transarc Corporation or Digital's distributed transaction manager (DECdtm) software. As a framework for DCE-based applications, the ACMSxp TP monitor simplifies application development, integrates system administration, and provides the additional capabilities of high availability, high performance, fault tolerance, and data integrity.

INTRODUCTION

Transaction processing (TP) is a style of computing that guarantees robustness and high availability for critical business applications. TP typically involves a large number of users using display devices to issue similar and repetitive requests. The requests result in the accessing and updating of one or more databases to reflect the current state of the business.

The basic building block in a TP system is a transaction. A transaction is an indivisible unit of work that represents the fundamental construct of recovery, consistency, and concurrency. Each transaction has the properties of atomicity, consistency, isolation, and durability (ACID). These properties are defined as follows:

- Atomicity. Either all the actions of a transaction succeed or all fail. In case of failure, the actions are rolled back.
- Consistency. After a transaction executes, it must either leave the system in a correct state or abort and return the system to its initial state.
- Isolation. The actions carried out by a transaction

against a shared database cannot become visible to other transactions until the transaction commits.

- Durability. The effects of a committed transaction are permanent.

A TP monitor manages and coordinates the flow of transactions through the system. Transaction requests typically originate from clients, are processed by one or more servers, and end at the originating client. When a transaction ends, the TP monitor ensures that all systems involved in the transaction are left in a consistent state.

The development of powerful desktop systems and advances in communications technology have fueled the growth of distributed client-server computing. The systems in a distributed environment may run different operating systems, possibly from different vendors. Business-critical applications may run under the control of different TP monitors. To coordinate their activities, TP monitors on heterogeneous systems need to conform to standards for open transaction processing.

Open standards for transaction processing have been adopted by the International Organization for Standardization/Open Systems Interconnection (ISO/OSI), the X/Open initiative, and the Service Providers' Integrated Requirements for Information Technology (SPIRIT) consortium.[1,2] The X/Open initiative is a consortium of vendors whose purpose is to define standards for application portability. SPIRIT is a consortium of telecommunications service providers from the U.S., Europe, and Japan working under the general sponsorship of the Network Management Forum (NMF).[3-5] The goal of the NMF's SPIRIT consortium is to define standards for portability and interoperability across heterogeneous systems to be used as the basis for procurement.

The main standards for open transaction processing are

- X/Open distributed transaction processing (DTP), which is an architecture that allows multiple programs to share resources (e.g., databases and files) provided by multiple resource managers and allows their work to be coordinated. The architecture defines application programming interfaces and interactions among transactional applications, transaction managers, resource managers, and communications resource managers. The transaction manager and the resource manager communicate by means of the XA interface.[6]

- X/Open transactional remote procedure call (TxRPC), which allows an application to invoke local and remote resource managers as if they were all local. TxRPC also allows an application to be decomposed into client and server components on different computers interconnected by means of remote procedure calls (RPCs).

- SPIRIT Structured Task Definition Language (STDL), which is a block-structured language for transaction processing.[4,5,7] STDL provides transactional features including demarcation of transaction boundaries, transaction recovery, exception handling, transactional communications, access to data queues, submission of queued work requests, and invocation of presentation services.

Digital's Application Control and Management System/ Cross-platform (ACMSxp) software product is a portable TP monitor that supports the open TP standards. It provides an environment for the development, execution, and administration of STDL applications. ACMSxp software is layered on the Open Software Foundation's (OSF's) Distributed Computing Environment (DCE) and supports multiple resource managers through Transarc Corporation's Encina toolkit on the UNIX operating system and Digital's distributed transaction manager (DECdtm) services on the OpenVMS operating system.[8] This paper describes the design of the ACMSxp TP monitor.

APPLICATION DEVELOPMENT

ACMSxp applications are written using a combination of the STDL and traditional languages such as C and COBOL. STDL is a modular, block-structured language developed specifically for transaction processing. It is based on the ACMS Task Definition Language (TDL) and was developed as part of Nippon Telegraph and Telephone's (NTT's) Multivendor Integration Architecture (MIA) initiative.[9-11] The NMF's SPIRIT consortium subsequently adopted STDL.

STDL Language Overview

STDL provides transactional features including transaction demarcation, transactional remote procedure call, transactional task and data record queuing, transactional display management, transactional exception handling, and transactional working storage called workspaces.

STDL divides an application into three parts: presentation, transaction flow control, and processing, as illustrated in Figure 1. The presentation part interfaces with display devices using a presentation manager, such as Motif, Windows, or forms manager software. The transaction flow control part is written in STDL and controls the flow of execution, including transaction demarcation, exception handling, and access to queues. The processing part is written in traditional languages, such as C, COBOL, and SQL, and provides computation and access to resource managers such as databases and files.

[Figure 1 (STDL Application Model) is not available in ASCII format.]

The application functions in the three parts of the STDL application model are referred to respectively as presentation procedures, tasks, and processing procedures. The application functions are packaged into groups for the purposes of compilation and execution. The groups are referred to as presentation groups, task groups, and processing groups.

A group specification describes the functions in the group and their interfaces. The interface specification includes the arguments that are passed to the function and an indication of whether an argument is input only, output only, or both input and output. For a task, the interface specification also indicates whether the task begins a new transaction (NONCOMPOSABLE) or joins the caller's transaction (COMPOSABLE).

STDL variables are defined in constructs called workspaces. Workspaces may have the transactional attribute, thus allowing an application to coordinate internal data with the outcome of the transaction along with other resource manager participants. Workspaces have the scope of either PRIVATE or SHARED. A PRIVATE workspace is accessible to only a single task; a SHARED workspace is accessible to all tasks in a task group.

STDL supports two types of queues: record and task. Record queues provide a transactional, durable scratch pad facility for applications to store and retrieve intermediate results. Task queues provide a way of executing tasks independently of the currently executing task in both time and location. Storage of the task queue element on the task queue may or may not be conditional on the outcome of the currently executing task.

Sample STDL Application

Figure 2 shows a sample STDL application program. The sample program accepts an integer, increments it, and displays it. In addition, shared workspaces are defined in the task group to track the number of successful executions ("successes") and the number of failed executions ("failures"). These operations all take place within the context of a transaction defined by task add1. If the transaction succeeds, the program increments "number" and the shared workspace "successes." If the transaction fails, the program restores "number" to its initial state and invokes the exception handler. The exception handler then updates the shared workspace "failures."

Figure 2 Sample STDL Application

```
RECORD arg1
      number INTEGER;
END RECORD;

TASK GROUP example1
      TASK add1
            TASK ARGUMENT IS arg1 PASSED AS INOUT;
END TASK GROUP;

TASK add1 ARGUMENT IS arg1 PASSED AS INOUT;
      WORKSPACES ARE       successes SHARED UPDATE RECOVERABLE,
                           failures SHARED UPDATE,
                           arg1 PRIVATE RECOVERABLE;
BLOCK WITH TRANSACTION
      PROCESSING
            COMPUTE successes = successes + 1
      PROCESSING
            COMPUTE number = number + 1
      EXCHANGE
            SEND RECORD number TO inscreen
END BLOCK
EXCEPTION HANDLER IS
      PROCESSING
            COMPUTE failures = failures + 1
END EXCEPTION HANDLER;
END TASK;
```

STDL Compiler

The STDL compiler simplifies the process of developing distributed client-server applications. It generates all the code necessary for supporting the application in the distributed environment, including server initialization, namespace registration, namespace lookup, and application context propagation. This allows the application programmer to focus on the application problem at hand.

The ACMSxp STDL compiler translates STDL specifications into executable code. The compiler itself is written in the ANSI C programming language using POSIX 1003.1 library interfaces for platform portability; the generated code consists of only ACMSxp run-time service calls and DCE service calls.[12] To the application programmer, the ACMSxp STDL compiler looks much like a classical compiler. The STDL compiler reads source code, converts it to object code, and then links it to create an executable program. Figure 3 shows the elements written by the application programmer and the transformations required to create an executable program.

[Figure 3 (STDL Compiler Flows) is not available in ASCII format.]

Internally, the STDL compiler consists of a series of steps that run under the control of a driver program. This processing takes place in the steps shown inside the dashed-line box of Figure 3. The STDL driver first reads STDL specifications in one pass and constructs internal structures that represent each STDL entity in the source file. Once an entity has been completely parsed and the syntax has been checked for errors, the driver generates intermediate files by translating

- STDL groups into ACMSxp client and server stubs and a DCE RPC Interface Definition Language (IDL) file
- STDL tasks into C code and ACMSxp run-time service calls
- STDL record definitions into C structures contained in C header files or COBOL copy files

After the STDL driver has generated all the intermediate files, it invokes the appropriate language processor to convert the files into object files. The DCE IDL compiler processes the IDL files, and the C compiler processes the tasks and the ACMSxp stubs. To keep the number of pieces visible to the application programmer within reason, the ACMSxp client and server stubs are combined with the DCE client and server stubs. The result is a collection of object files similar to those found in a conventional DCE application. The platform linker then combines the resulting files into an executable program.

The ACMSxp client and server stubs are similar in concept to the DCE RPC client and server stubs. The client stub is linked with other applications that invoke this group's tasks or procedures. The server stub is combined with application code to create the application server image. The ACMSxp stubs call ACMSxp run-time services to add to the base DCE RPC services features such as transactions, failover and failback, and time-out.

EXECUTION ENVIRONMENT

The ACMSxp run-time system provides an environment for executing and invoking STDL applications. It also provides services that allow components in the execution environment to be managed. The execution environment provides many services typically needed in TP environments, such as resource scheduling, fault tolerance, and queuing.

Process Model

The ACMSxp environment consists of client and server components. A TPsystem comprises multiple server components on a node that are managed as a unit. A given TPsystem has a globally unique name and is associated with only one node, but a node can have multiple TPsystems associated with it. A TPsystem contains a central process called the TPcontroller, which controls the components within the TPsystem. The processes in the execution environment are illustrated in Figure 4.

[Figure 4 (Processes in Execution Environment) is not available in ASCII format.]

As the central point of control for the components within a TPsystem, the TPcontroller performs many functions, including license checking, starting and stopping servers, and monitoring server processes and restarting them when they terminate abnormally. It also receives administration requests and performs the requested operations, maintains information in shared memory for communication with server processes, and maintains key files for server authentication.

A task server executes STDL task group code and uses multiple threads in a single process to achieve concurrent execution (multithreaded). A processing server executes STDL processing group code and uses a pool of single-threaded processes to achieve concurrent execution (multiprocess).

System servers provide specific run-time services to the TPcontroller, task servers, and processing servers. The system servers include the event log server, the request queue server, and the record queue server. System servers are multithreaded.

Client processes invoke services provided by a TPsystem and its servers. An administration client (also referred to as the director) invokes administration services provided by the TPcontroller and system servers. An application client invokes application services provided by task servers. An application client can be a customer-written client or an ACMS Desktop client. A customer-written client can consist of code necessary to support a forms manager or device control such as an automatic teller machine or a gas pump. An ACMS Desktop client allows popular desktop systems such as the Macintosh, SCO's UNIX, Microsoft Windows, and Windows NT operating systems to be used to access services provided by ACMSxp application servers.

Run-time Services

The ACMSxp run-time system provides services required for the execution of client-server TP applications. The run-time services are highly modular and are layered on the services provided by the underlying transaction manager, DCE, operating system, network, and other services, as shown in Figure 5.

[Figure 5 (Modular Run-time Architecture) is not available in ASCII format.]

The run-time services integrate the services of the underlying platform and provide additional functionality. They export an application programming interface (API) called the transaction processing service interface (TPSI). The run-time services include

- Communication, which provides services for transactional and nontransactional communication between clients and servers using DCE RPC. The supported transports are transmission control protocol/internet protocol (TCP/IP), DECnet OSI, and Fast Local Transport.

- Process management, which provides services for starting and stopping server processes, monitoring server processes for abnormal termination, and restarting new ones to maintain the specified number of processes.

- Thread context management, which provides services for creating, setting, and propagating thread context. Thread context includes request context, exception context, transaction context, and procedure context.

- Timer alert, which provides services for accumulating CPU time and transaction (elapsed) time.

- Transaction demarcation, which integrates with the Encina toolkit on the OSF/1 platform or the DECdtm software on the OpenVMS platform to provide distributed transaction support.

- Queuing, which provides services for request queuing and record queuing. Request queuing allows task requests to be queued for deferred invocation. Record queuing allows data records to be enqueued and dequeued.

- File management, which provides file management services for COBOL and C programs. It provides thread-based transaction semantics for STDL file access and handles opening and closing of files, file positioning, and file locking.

- Workspace management, which provides services for managing private and shared workspaces. A workspace is an STDL construct and represents an area of memory used for data storage and for arguments passed in a procedure call. A workspace can be recoverable or nonrecoverable.

- Security, which authenticates users and servers and provides access control, based on the DCE security service, for application invocation as well as management operations.

- Event posting, which provides services for writing events into a log. Logged events include error, security, status, audit, and trace events.

- Performance monitoring, which provides services for capturing performance measurement data.

Client-Server Communication

The ACMSxp communications services use OSF's DCE services for locating servers, invoking servers, and ensuring secure communications. The communications services maximize the efficiency of DCE service usage, provide robustness in the event of failure, and add distribution of transaction semantics to DCE RPC communications.

Figure 6 shows the elements and steps involved in the communication between a client and a server. The numeric annotations in the following discussion refer to the numbers that appear in the figure.

[Figure 6 (Client-Server Communication Flow) is not available in ASCII format.]

The STDL client application calls the server (1). The ACMSxp client stub issues run-time service calls (2) to initialize context blocks and to obtain a binding handle (i.e., server addressing information), and calls the DCE RPC client stub, passing context blocks and application data (3). The DCE RPC client stub marshals data and calls the server (4).

The DCE RPC server stub receives the call, unmarshals data, and calls the ACMSxp server stub (5). The ACMSxp server stub issues run-time service calls (6) to establish local context and to check security authorization, and calls the server application (7). The server application executes and returns the results to the ACMSxp server stub, which propagates any error information.

Transaction Processing Characteristics

The run-time system provides the TP monitor with characteristics such as high availability, load balancing, and high performance. Some of the mechanisms used to achieve these characteristics are discussed below.

Availability. The run-time system provides failover and failback capabilities to enhance the availability of applications. Failover is the redirection of an RPC to an alternate server if the intended server is not reachable. The target server can be unreachable for many reasons, including loss of connectivity, application failures, and machine failures. Failback is the redirection of calls to the original server when it becomes available.

Failover and failback capabilities are supported for task servers but not for processing servers. The DCE cell directory service (CDS) namespace profile mechanism supports failover and failback. The system administrator configures the primary and alternate servers by placing them in the same namespace profile with different priorities. The server with the lower priority number is the primary server.

Run-time support for failover and failback is implemented in the client stub. Failover is attempted if an RPC fails and the returned error indicates that no work had been done by the called server in the current transaction. Failover is always attempted for a nontransactional RPC but is attempted for a transactional RPC only if this is the first call to the intended server in the transaction. The failover mechanism is optimized in three ways: by reconnecting, by pinging, and by checking the failed servers table. When a failure is detected, the failover mechanism attempts to reconnect to the server in case the failure was caused by intermittent communications problems. If the reconnect fails, the failover mechanism attempts to find an alternate server. When an alternate server is selected, it is pinged to ensure that it is reachable before being called with application work. If a server cannot be reached, it is recorded in a "failed servers" table and skipped on subsequent failover attempts.

Failback is attempted if the binding found is for an alternate server. Failback to the primary server is attempted even if the binding for the alternate server is good, as long as the failback

timer has expired. The failback timer defaults to 300 seconds and can be set by an environment variable.

Load Balancing. The ACMSxp run-time system can achieve load balancing for task servers through the DCE CDS. The DCE CDS group entry contains multiple server entries that provide the same interface. Locating a server by means of a group entry results in the random selection of one server in the entry. A combination of static load balancing and failover can also be implemented using DCE CDS functionality.

Performance. Many parts of the ACMSxp system contain mechanisms that are designed to improve performance. A discussion of some of these mechanisms follows.

The server stub caches server bindings to improve performance. Server bindings are the addressing information that allows a client process to call a server process. Binding caching is a means of retaining the server addressing information for reuse. Reading the binding from the namespace can be time-consuming. For example, a DCE CDS namespace lookup requires a network connection to fetch the data from another process, which may be on a separate node. The cache of server bindings is shared among all the threads in the client process. This sharing provides a second order of performance improvement in that work previously performed on behalf of other threads can improve the performance of all threads by preloading the cache.

The scheduler subcomponent of the communications services allocates and deallocates server processes. It maintains a local namespace (also referred to as scheduler database) in shared memory to keep track of server process allocation. The use of the local namespace instead of DCE CDS improves the performance of RPC calls between task servers and processing servers, which are required to be in the same TPsystem.

The security service caches access control lists (ACLs) to improve performance. The TPcontroller maintains in shared memory the ACLs for managed objects that the ACMSxp TP monitor accesses at run time (e.g., procedures). The security service caches each object's ACL into the server process memory when the object is first accessed. The server process refreshes its cache if the entry in shared memory is updated.

SYSTEM ADMINISTRATION

The distributed TP environment is inherently complex and requires effective system administration. The ACMSxp TP monitor provides the following system administration facilities for configuring, monitoring, and controlling components and resources within the ACMSxp run-time environment:

- Integrated user interface. The director (see the discussion of Figure 7, which follows) provides a consistent user interface for invoking management operations on all managed objects. The command line interface provides features such as command scripts, symbol substitution, session logging, default session parameters, and on-line help.

- Centralized distributed management. A single director can manage multiple TPsystems on the local or remote nodes using DCE RPC for communication.

- Extensibility. The object-oriented approach allows the ACMSxp TP monitor to represent managed resources in a consistent manner and to add new objects gracefully.

Management Model

The ACMSxp management model is object oriented and is based on the ISO/OSI standard for network and system management.[2,13] Figure 7 illustrates the elements of the model.

[Figure 7 (Management Model) is not available in ASCII format.]

A director initiates management requests on behalf of the system administrator and serves as the interface between a system administrator and the objects being monitored and controlled.[14] A director consists of two parts: the user interface and the management service interface. The user interface interacts with the user and is either command line or graphical. The management service interface interacts with management agents. This interface provides services for creating an association for communication between a director and management agents, for initiating management requests, for returning results to the director, for canceling an outstanding request without waiting for completion, and for terminating an association normally.

The management protocol specifies both the mechanism for communication between a director and management agents and the model of interaction between them. The model specifies how requests and responses are passed between the director and the management agents, the processing of requests that involve wild card object instances, and the buffering of multiple responses to optimize performance. The ACMSxp TP monitor uses DCE RPC for communication between a director and management agents.

A management agent performs operations for a managed object. Each object class has a management agent that performs management operations for instances of that object class. The management agent receives a management request from the director, performs the requested operation, and returns the results.

Management Functions

Management operations that can be performed on managed objects are grouped into the following functional categories, as defined by the OSI management framework:

- Configuration management. Managed objects are instantiated, observed, and controlled. Persistent information about managed objects is stored in a configuration database.

- Fault management. Events generated by system operation are recorded in a log file. The contents of the log can be examined using a variety of search criteria.

- Performance management. Performance metrics are collected as the run-time system executes. The performance data is captured as attributes of managed objects and can be examined using the director.

- Security management. Principals are authenticated using the DCE. Access to system administration operations and application procedures is controlled using an ACL mechanism based on the DCE model.

Managed Objects

The resources in the ACMSxp environment that need to be managed are represented as objects. A managed object encapsulates the functionality of a real resource and specifies as visible only those aspects that need to be accessed by the manager. A managed object has the following properties:

- Attributes. Attributes are pieces of information that describe an object and represent internal state variables. Each attribute has a name and a value, which can be examined or modified as a result of a management operation. Examples of attributes are executable file name and processing state (for a server).

- Operations. Operations are activities that the manager can perform on the managed object. Operations allow the manager to examine attributes, modify attributes, and perform actions specific to the object. Examples of operations are create, delete, enable, disable, set, and show.

- Events. Events indicate the occurrence of normal and abnormal conditions. Examples of events are the detection of an error and the arrival at a threshold.

- Behavior. Behavior defines how attributes, operations, and events work together and how they affect the managed resource.

For naming purposes, managed objects are organized into a containment hierarchy. This hierarchy specifies which managed object is contained within another and reflects the containment relationship of all their corresponding managed resources. The top-level object in the structure, referred to as a global object, has a globally unique name. Objects contained within the global object are referred to as local objects and have names that are unique only within the context of their level in the structure. Table 1 describes the managed objects in the ACMSxp system.

Table 1 ACMSxp Managed Objects

Object Class	Description
TPsystem	A collection of system and application components and resources on a given node that is managed as a unit. A TPsystem is referred to as a global entity because it contains other managed objects and is not contained in any other managed object.
Server	A managed object that executes procedures. It encapsulates a collection of one or more operating system processes that execute the same program image.
Process	The basic unit scheduled by the operating system that provides the context within which a program image executes. It represents an operating system process.
Interface	A set of procedures that is provided by a server. It represents a DCE RPC interface and has a universally unique identifier (UUID) that distinguishes it from other instances.
Procedure	A structured sequence of instructions executed to achieve a particular result. It represents a DCE RPC operation.
Queue	A repository for storing an ordered collection of elements. The supported queues include a request queue, which contains submit requests, and a record queue, which contains data records.
Element	A single entry in a queue.
Log	A named repository where event records are stored.
Request session	The occurrence of a request at a particular TPsystem. A request is a series of operations invoked by a client program on behalf of a user and executed by one or more servers.

CONCLUSION

The ACMSxp transaction processing monitor employs modular design techniques and a proven transaction processing architecture to provide a truly open, distributed transaction processing system. The STDL application development language, which the ACMSxp TP monitor supports, has been endorsed by an international standards consortium and has been implemented on other vendors' platforms. The layering on both the Open Software Foundation's Distributed Computing Environment software and the Encina toolkit provides a foundation of open distributed processing that has been accepted by the world's largest computer systems providers. The ACMSxp TP monitor provides a comprehensive set of facilities for managing the run-time environment. The object-oriented management approach results in a consistent representation of managed objects, a consistent user interface, a modular implementation, and extensibility.

ACKNOWLEDGMENTS

Throughout the course of this project, many people have participated in the design, implementation, and documentation of the product. The authors would like to thank all these people for their dedication and their contributions.

REFERENCES

1. X/Open Distributed Transaction Processing Reference Model, ISBN 1-872630-16-2 (Reading, U.K.: X/Open Company Ltd., 1991).

2. Information Processing Systems -- Open Systems Interconnection --Basic Reference Model -- Part 4: Management Framework, ISO/IEC 7498-4:1989 (Geneva: International Organization for Standardization, 1989).

3. SPIRIT Platform Blueprint (SPIRIT 2.0, vol. 1), ISBN 1-85912-059-8, Document No. J401 (Reading, U.K.: X/Open Company Ltd., 1994).

4. SPIRIT STDL Language Specification (SPIRIT 2.0, vol. 3), ISBN 1-85912-063-6, Document No. J403 (Reading, U.K.: X/Open Company Ltd., 1994).

5. SPIRIT STDL Environment, Execution and Protocol Mapping (SPIRIT 2.0, vol. 4), ISBN 1-85912-064-4, Document No. J404 (Reading, U.K.: X/Open Company Ltd., 1994).

6. X/Open CAE Specification, December 1991, Distributed

Transaction Processing: The XA Specification, ISBN 1-872630-24-3, Document No. C193 or XO/CAE/91/300 (Reading, U.K.: X/Open Company Ltd., 1994).

7. P. Bernstein, P. Gyllstrom, and T. Wimberg, "STDL---Portable Language for Transaction Processing," Proceedings of the Nineteenth International Conference on Very Large Databases, Dublin, Ireland, 1993.

8. W. Laing, J. Johnson, and R. Landau, "Transaction Management Support in the VMS Operating System Kernel," Digital Technical Journal, vol. 3, no. 1 (Winter 1991): 33-44.

9. T. Speer and M. Storm, "Digital's Transaction Processing Monitors," Digital Technical Journal, vol. 3, no. 1 (Winter 1994): 18-32.

10. Multivendor Integration Architecture, Division 1, Overview, Technical Requirements, TR550001 (Tokyo, Japan: Nippon Telegraph and Telephone Corporation, 1991).

11. E. Newcomer, "Pioneering Distributed Transaction Management," Bulletin of the Technical Committee on Data Engineering, vol. 17, no. 1 (March 1994).

12. Information Technology -- Portable Operating System Interface (POSIX) -- Part 1: System Application Interface (API) [C Language], IEEE 1003.1-1990 (New York: The Institute of Electrical and Electronics Engineers, 1990).

13. M. Sylor, F. Dolan, and D. Shurtleff, "Network Management," Digital Technical Journal, vol. 5, no. 1 (Winter 1993): 117-129.

14. C. Strutt and J. Swist, "Design of the DECmcc Management Director," Digital Technical Journal, vol. 5, no. 1 (Winter 1993): 130-142.

BIOGRAPHIES

Robert K. Baafi

A principal software engineer in the Transaction Processing Engineering Group, Robert Baafi is the primary architect for the system management component of the ACMSxp transaction processing monitor. Prior to joining Digital in 1989, he was the project leader for Cullinet Software's IDMS-DC transaction processing monitor. Bob received a B.S. in electrical engineering from the University of Connecticut in 1971 and an M.S. in information systems from Lehigh University in 1973. He is a member of ACM, Tau Beta Pi, and Eta Kappa Nu.

J. Ian Carrie

Ian Carrie is the project leader for the ACMS Desktop product. He is a member of the Transaction Processing Engineering Group. Since joining Digital in 1989, Ian has also contributed to the ACMSxp transaction processing monitor. He worked on the STDL compiler code generator, language run-time support, file support, and Encina transaction manager integration. In earlier work, he was employed by Cullinet Software as a project leader in the IDMS/R database product's Communications Group. Ian holds a B.A. (1980) in computer science/managerial studies from Rice University. He is a member of ACM.

William B. Drury

Bill Drury is currently employed by Stratus Computer as Engineering Manager, Transaction Processing and System Performance. While a consulting engineer at Digital, he led the design and development of the ACMSxp transaction processing monitor. He presented the product at numerous technical forums, including the STDL Implementors' Workshop, DECUS, DECORUM '94 (Transarc Corporation's user group), and the OSF DCE Developers' Conference. He also contributed to the specification of the Multivendor Integration Architecture (MIA) on which the ACMSxp product is based. Bill received B.S.E.E. (1982) and M.S.E.E. (1986) degrees from Ohio University.

Oren L. Wiesler

Presently, Oren Wiesler is a Factory Integration Manager at PRI Automation, a manufacturer of automation equipment used in semiconductor manufacturing. At Digital, he was a principal software engineer in the Transaction Processing Engineering Group. He led the ACMSxp for OSF/1 AXP version 2.0 effort and the

support team for ACMSxp for OpenVMS VAX version 1.0, and contributed to the ACMSxp run-time system. Earlier, Oren worked in a processor hardware group. He received a B.S.E.E. from Worcester Polytechnic Institute in 1984 and holds two patents: one related to dynamic control of simultaneously switching outputs, the other on interleaved control store addressing.

TRADEMARKS

==========

==========