Digital's Transaction Processing Monitors

By Thomas G. Speer and Mark W. Storm

Abstract

Digital provides two transaction processing (TP) monitor products- ACMS (Application Control and Management System) and DECintact (Integrated Application Control). Each monitor is a unified set of transaction processing services for the application environment. These services are layered on the VMS operating system. Although there is a large functional overlap between the two, both products achieve similar goals by means of some significantly different implementation strategies. Flow control and multithreading in the ACMS monitor is managed by means of a fourth-generation language (4GL) task definition language. Flow control and multithreading in the DECintact monitor is managed at the application level by third-generation language (3GL) calls to a library of services. The ACMS monitor supports a monitors will be their different application programming interfaces.

Introduction

Transaction processing is the execution of an application that performs an administrative function by accessing a shared database. Within transaction processing, processing monitors provide the software "glue" that ties together many software components into a transaction processing system solution.

A typical transaction processing application involves interaction with many terminal users by means of a presentation manager or forms system to collect user requests. Information gathered by the presentation manager is then used to query or update one or more databases that reflect the current state of the business. A characteristic of transaction processing systems and applications is many users performing deferred task model of queuing, and the DECintact monitor supports a message-based model. Over time, the persistent distinguishing feature between the two

a small number of similar functions against a common database. A transaction processing monitor is a system environment that supports the efficient development, execution, and management of such applications.

Processing monitors are usually built on top of or as extensions to the operating system and other products such as database systems and presentation services. By so doing, additional components can be integrated into a system and can fill "holes" by providing functions that are specifically needed by transaction processing applications. Some examples of these functions are application control and management, transaction-processing-specific execution environments, and transaction-processing-specific programming interfaces.

Digital provides two transaction processing monitors: the Application Control and Management System (ACMS) and the DECintact monitor. Both monitors are built on top of the VMS operating system. Each monitor provides a unified set of transaction-processing-specific services to the application environment, and a large functional overlap exists between the services each monitor provides. The distinguishing factor between the two monitors is customers have their own styles of application programming. Those that prefer 4GL styles should be able to build transaction processing applications using Digital's TP monitors without changing their style. Similarly, those that prefer 3GL styles should also be able to build TP applications using Digital's TP monitors without changing their style.

The ACMS monitor was first introduced by Digital in 1984. The ACMS monitor addresses the requirements of large, complex transaction processing applications by making them easier to develop and manage. The ACMS monitor also creates an efficient execution environment for these applications.

The DECintact monitor (Integrated Application Control) was originally developed by a third-party vendor. Purchased and introduced by Digital in 1988, and it has been installed in major financial institutions and manufacturing sites. The DECintact monitor includes its own presentation manager, support for DECforms, a recoverable queuing subsystem, a transaction manager, and in the area of application programming styles and interfaces-fourth-generation language (4GL) versus third-generation language (3GL). This distinction represents Digital's recognition that

a resource manager that provides its own recovery of RMS (Record Management Services) files.

This paper highlights the important similarities and differences of the ACMS and DECintact monitors in terms of goals and implementation strategies.

Development Environment

Transaction processing monitors provide a view of the transaction processing system for application development. Therefore, the ACMS and DECintact monitors must embody a style of program development.

ACMS Programming Style

A "divide and conquer" approach was used in the ACMS monitor. The work typically involved in developing a TP application was divided into logically separate functions described below. Each of these functions was then "conquered" by a special utility or approach.

In the ACMS monitor, an "application" is defined as a collection of selectable units of work called tasks. A separate application definition facility isolates the system management characteristics of the application (such as resource allocation, file location, and protection) from the logic of the application. A separate menu definition facility lists entries that a user can select. Each entry points either to another menu or to an application and a task. (Decoupling menus from the application allows user menus to be independent of how the tasks are grouped into applications.)

In addition to separate menu specification and system management characteristics, the application logic is broken down into the three logical parts of interactive TP applications:

- o Exchange steps support the exchange of data with the end user. This exchange is typically accomplished by displaying a form on a terminal screen and collecting the input.
- o Processing steps perform computational processing and database or file I/O through standard subroutines. The subroutines are written in any language that accepts records passed by reference.
- o The task definition language defines the flow of control between processing steps and exchange steps and specifies transaction demarcation. Work spaces are special

The specification of menus is also decoupled from the application. A nonprocedural (4GL) method of defining menu layouts is used in which

the layouts are compiled into form files and data structures to be used at run-time. Each menu entry

records that the ACMS monitor provides to pass data between the task definition, exchange steps, and processing steps.

A compiler, called the application definition utility (ADU), is implemented in the ACMS monitor to compile the task definition language into binary data structures. The run-time system is table-driven, rather than interpreted, by these structures.

Digital is the only vendor that supplies this "divide and conquer" solution to building large complex TP applications. We believe this approach-unique in the industry-reduces complexity, thus making applications easier to produce and to manage.

DECintact Programming Style

The approach to application development used in the DECintact monitor provides the application developer with 3GL control over the transaction processing services required. This approach allows application prototyping and development to be done rapidly. Moreover, the application can make the most efficient use of monitor services by selecting and controlling only those services required for a particular task.

In the DECintact monitor, an application is defined as one or more programs written entirely in 3GL DECintact monitor. All DECintact services are callable, including most services provided by the DECintact utilities. The DECintact services are as follows:

o A library of presentation services used for all interaction with users. The application developer includes calls to these services for form manipulation and display. Forms are created with a forms editor utility and can be updated dynamically. Forms are displayed by the DECintact terminal manager in emulated block mode. Device- and terminal-dependent information is completely separated from the implementation of the application.
o The separation of specification of menus from the application. DECintact menus are defined by means of a menu database and are compiled into data structures accessed at run-time. The menus are tree-structured. Each entry points either to another menu entry or to an executable application image. The specification of menus is linked to the

and supported by the VMS system. The code written by the application developer manages all flow control, user interaction, and data manipulation through the utilities and service

libraries provided by the

DECintact monitor's security subsystem. The DECintact terminal user sees only those specific menu entries for which the user has been granted access.

- A library of services for the control of file and queue operations. In addition to layered access to the RMS file system, the DECintact monitor supports its own hash file format (a functional analog to single-keyed indexed files in RMS) which provides very fast, efficient record retrieval. The application developer includes calls to these services for managing RMS and hash file I/O operations, demarcating recovery unit boundaries, creating queues, placing data items on queues, and removing data items from queues. The queuing subsystem is typically an integral part of application design and work flow control. Application-defined DECintact recovery units ensure that RMS, hash, and queue operations can be committed or aborted atomically; that is, either all permanent effects of the recovery unit happen, or none happen.

Because of DECintact's 3GL development environment, application programmers who are accustomed to calling procedure libraries from 3GL are required. Further, completed applications can be produced quickly because training time is minimal.

On-line Execution Environment

Transaction processing monitors provide an execution environment tailored to the characteristics and needs of transaction processing applications. This environment generally has two aspects: on-line, for interactive applications that use terminals; and off-line, for noninteractive applications that use other devices.

Traditional VMS timesharing applications are implemented by allocating one VMS process to each terminal user when the user logs in to the system. An image activation is then done each time the terminal user invokes a new function. This method is most beneficial in simple transaction processing applications that have a relatively small number of users. However, as the number of users grows or as the application becomes larger and more complex, several problem areas may arise with this method:

- Resource use. As the number of processes standard VMS languages or who are familiar with other transaction processing monitors can easily learn DECintact's services. Application prototypes can be produced quickly because only skills in

grows, more and more memory is needed to run the system effectively.

- Start-up costs. Process creation, image activation, file opens, and database binds are expensive operations in terms of system resources utilized and time elapsed. These operations can degrade system performance if done frequently.
- Contention. As the number of users simultaneously accessing a database or file grows, contention for locks also increases. For many applications, lock contention is a significant factor in throughput.
- Processing location. Single process implementations limit distribution options.

ACMS On-line Execution

To address the problems listed above, Digital implemented a client/server architecture in the ACMS monitor. (Client/server is also called request /response.) The basic run-time architecture consists of three types of processes, as shown in Figure 1: the command process, execution controller, and procedure servers.

the command process implements the functions of a request initiator, presentation manager, and request manager for direct requests.) [1] The command process is generally created at system start-up time, although ACMS commands allow it to be started at other times. The process is multithreaded through the use of VMS asynchronous system traps (AST). Thus, one command process per node is generally sufficient for all terminals handled by that node.

There are two subcomponents of the ACMS monitor within the command process:

- System interface, which is a set of services for submitting work requests and for interacting with the ACMS application
- DECforms, Digital's forms management product, which implements the ANSI /ISO Forms Interface Management System (FIMS) that provides the presentation server for executing the exchange steps

An agent in the ACMS monitor is a process that submits work requests to an application. In the ACMS system, the command process is a special agent responsible for interactions with the terminal user. (In terms of the DECdta architecture, The command process reads the menu definition for a particular terminal user and determines which menu to display. When the terminal user selects a particular menu entry, the command process calls the ACMS system interface services to submit the task. The system interface uses logical names from the VMS system to translate the application name into the address of the execution controller that represents that application. The system interface then sends a message to the execution controller. The message contains the locations of the presentation server and an index into the task definition tables for the particular task. The status of the task is returned in the response. During the course of task execution, the command process accepts callbacks from the task to display a form for interaction with the terminal user.

The execution controller executes the task definition language and creates and manages procedure servers. The controller is created at application start-up time and is multithreaded by using VMS ASTs. There is one execution controller per application. (In terms

When the execution controller receives a request from the command process, it invokes DECdtm (Digital Distributed Transaction Manager) services to join the transaction if the agent passes the transaction identifier. If the agent does not pass a transaction identifier, there is no transaction to join and a DECdtm or resource-manager-specific transaction is started as specified in the task definition. The execution controller then uses the task index to find the tables that represent the task. When the execution of a task reaches an exchange step, the execution controller sends a callback to the command process for a form to be displayed and the input to be collected for the task. When the request to display a form is sent to the command process, the execution controller dismisses the AST to enable other threads to execute. When the response to the request arrives from the exchange step, an AST is added to the queue for the execution controller.

When a task comes to a processing step, the execution controller allocates a free procedure server to the task. It then sends a request to of the DECdta architecture, the execution controller and the procedure servers implement the functions of a transaction server.) [1]

the procedure server to execute the particular procedure and dismisses the AST. If no procedure server is free, the

execution controller puts the request on a waiting list and dismisses the AST. When a procedure server becomes free, the execution controller checks the wait list and allocates the procedure server to the next task, if any, on the wait list.

Procedure servers are created and deleted by the execution controller. Procedure servers are a collection of user-written procedures that perform computation and provide database or file accesses for the application. The procedures are written in standard languages and use no special services. The ACMS system creates a transfer vector from the server definition. This transfer vector is linked into the server image. With this vector, the ACMS system code can receive incoming messages and translate them into calls to the procedure.

A procedure server is specified with initialization and termination procedures, which are routines supplied by the user. The ACMS monitor calls these procedures whenever a procedure server is created and deleted. The initialization procedure opens files and performs database bind operations. The termination procedure

The ACMS architecture addresses the problem areas discussed in the On-line Execution Environment section in several ways.

Resource Use. Because procedure servers are allocated only for the time required to execute a processing step, the servers are available for other use while a terminal user types in data for the form. Thus, the system can execute efficiently with fewer procedure servers than active terminal users. Improvement gains in resource use can vary, depending on the application. Our debit and credit benchmark experiments with the ACMS monitor and the Rdb/VMS relational database system indicated that the most improvement occurs with one procedure server for every one or two transactions per second (TPS). These benchmarks equate to 1 procedure server for every 10 to 20 active terminal users.

The use of procedure servers and the multithreaded character of the execution controller and the command process allow the architecture to reduce the number of processes and, therefore, the number of resources needed. The optimal does clean-up work, such as closing files prior to process exit.

solution for resource use would consist of one large multithreaded process that performed all processing. However, we chose to trade off some resource use in the architecture in favor of other gains.

- Ease of use-Multithreaded applications are generally more difficult to code than single-threaded applications. For this reason, procedure server subroutines in the ACMS system can be written in a standard fashion by using standard calls to Rdb/VMS and the VMS system.
- Error isolation-In one large multithreaded process, the threads are not completely protected within the process. An application logic error in one thread can corrupt data in a thread that is executing for a different user. A severe error in one thread could potentially bring down the entire application. The multithreaded processes in the ACMS architecture (i.e., the execution controller and command process) are provided by Digital. Because no application code executes directly in these processes, we can guarantee that no application coding error can affect them. Procedure servers are single-threaded. Therefore, an application logic

Start-up Costs. The run-time environment is basically "static," which means that the start-up costs (i.e., system resources and elapsed time) are incurred infrequently (i.e., at system and application start-up time). A timesharing user who is running many different applications causes image activations and rundowns by switching among images. Because the terminal user in the ACMS system is separated from the applications processes, the process of switching applications involves only changing message destinations and incurs minimal overhead.

Contention. The database accesses in the ACMS environment are channeled through a relatively few, but heavily used, number of processes. The typical VMS timesharing environment uses a large number of lightly used processes. By reducing the number of processes that access the database, the contention for locks is reduced.

Processing Location. Because the ACMS monitor is a multiprocess architecture, the command process and forms processing can be done close to the terminal user on small, inexpensive error in a procedure server is isolated to affect only the task that is executing in the procedure server.

machines. This method takes advantage of the inexpensive processing power available on these smaller machines while the rest of the application executes on a larger VAXcluster system.

DECintact On-line Execution

Although the specific components of the DECintact monitor vary from those of the ACMS monitor, the basic architecture is very similar. Figure 2 shows the application configured locally to the front end. The run-time architecture consists of three types of DECintact system processes-terminal manager /dispatcher, DECforms servers, server manager- and, typically, one or more application processes. When forms processing is distributed, the same application is configured as shown in Figure 3.

The DECintact monitor can run in multiple copies on any one VAX node. Each copy can be an independent run-time environment; or it can share data and resources, such as user security profiles and menu definitions, with other copies on the same system. Thus, independent development, testing, and production environments can reside on the same node.

In the DECintact system, the terminal manager /dispatcher process (one per copy) is responsible for the following:

- o Displaying DECintact forms
- o Coordinating DECforms forms display
- o Interacting with local applications
- o Communicating, through DECnet, with remote DECintact copies
- o Maintaining security authorization, including

Applications designated in the local menu database as remote applications cause the front-end terminal manager/dispatcher process to communicate with the cooperating back-end terminal manager/dispatcher process through a task-to-task DECnet link. (In terms of the DECdta architecture, the terminal manager /dispatcher implements the functions of presentation manager, request initiator, and request manager for direct requests.) [1]

When a user selects the remote task, that user's request is sent to the back end and is treated by the application as a local request. The terminal manager/dispatcher process is started automatically as part of a copy start-up and is multithreaded. Therefore, one such process can handle all the terminal users for a particular DECintact copy.

When the terminal user

the dynamic generation
of user-specific menus

selects a menu task, one
of the following actions

occurs, depending on
whether the task is local
or remote and whether it is
single- or multithreaded.

If the application is local and single-threaded, a VMS process may be created that activates the application image associated with this task. The terminal manager /dispatcher, upon start up, may create a user-specified number of application shell VMS processes to activate subsequent application images. If such a shell exists when the user selects a task, this process is used to run the application image. Each user who selects a given menu entry receives an individual VMS process and image.

If the application is local and multithreaded, the terminal manager/dispatcher first determines whether this task has already been activated by previous users. If the task has not been activated and a shell is not available, the terminal manager/dispatcher creates a VMS process for the application and activates the image. If the task is already activated, the terminal manager /dispatcher connects the user to the active task. The user becomes another thread of execution within the image. Multithreaded applications handle many simultaneous users within the context of one VMS process and image.

/dispatcher processes the selection locally by using the same procedures as described above.

Local DECintact forms interaction is handled in the following manner by the local terminal manager/dispatcher. The application's call to display a form sends a request to the terminal manager. The terminal manager locates the form in its database of active forms, displays the form on the user's terminal, and returns control to the application when the user has entered all data in the form. If the application is remote, form information is sent between cooperating local and remote terminal manager processes; the interface is transparent to the application.

In addition to supporting DECintact forms, the DECintact monitor also supports applications that use DECforms as their presentation service. The implementation of this support follows the same client/server model used by the ACMS system's support for DECforms and shares much of the underlying run-time interprocess communication code used by the ACMS monitor. Functionally, the two implementations of DECforms

Remote applications, whether single- or multithreaded, route the menu task selection to a remote terminal manager /dispatcher process. On receipt of the request, the remote terminal manager

support are also similar to the ACMS monitor. Both implementations offer transparent support for distributed DECforms processing, automatic forms caching (i.e., propagation of updated DECforms in a distributed environment), and DECforms session caching for increased performance.

The DECintact monitor supports application-level, single- and multithreaded environments. The DECintact monitor's threading package allows application programmers to use standard languages supported by the VMS system to write multithreaded processes. Applications declare themselves as either single- or multithreaded. With the exception of the declaration, there is little difference between the way an on-line multithreaded application and its single-threaded counterpart must be coded. For on-line applications, thread creation, deletion, and management are automatic. New threads are created when a terminal user selects the multithreaded application and are deleted when the user leaves the application.

In a single-threaded application, the following occurs:

o Each user receives an individual VMS process and image context (e.g., 200 users, 200 processes).
o All terminal and file

In a multithreaded on-line application, the following occurs:

o One VMS process/image can handle many simultaneous users.
o All terminal and file I/O is asynchronous.
o New threads are created automatically when new users are connected to the process.
o The application image does not exit when all currently allocated threads have completed execution but remains for use by new on-line users.

For each thread in a multithreaded application image, the DECintact system maintains thread context and state information. Each I/O request is issued asynchronously. Immediately after control is returned, but before the I/O request completes, the DECintact system saves the currently executing thread's context and schedules another thread to execute. When the thread's I/O completion AST is delivered, the thread's context is restored, and the thread is inserted on an internally maintained list of threads eligible for execution.

A thread's context consists of the following:

I/O is synchronous.

- The application image normally exits when the application work is completed.

- An internally maintained thread block containing state information
- The stack

- Standard DECintact work spaces that are allocated to each thread and that maintain terminal and file management context

- Local storage (e.g., the $LOCAL PSECT in COBOL applications) that the application has designated as thread-specific

The PSECT naming convention allows the application to decide which variable storage is thread-specific and which is process-global. Thread-specific storage is unavailable to other threads in the same process because it is saved and restored on each thread switch. Process-global storage is always available to all threads in the process and can be used when interthread communication or synchronization is desired.

The use of multithreading in the DECintact system is appropriate for higher volume multiuser applications that perform frequent I/O. Such application usage is typical in transaction processing environments. Because thread switches occur only when I/O is requested or when locking programs or other batch-oriented processing. These kinds of applications typically choose to declare themselves as single-threaded.

All I/O from within a multithreaded DECintact application process is asynchronous. Therefore, the DECintact system provides a client/server

interface between multithreaded applications and synchronous database systems, such as VAX DBMS (Database Management System) and Rdb/VMS systems. The interface is provided because calling a synchronous database operation directly from within a multithreaded application would stall the calling thread and all other threads until the call completed. Figure 2 shows that a typical on-line DECintact application accessing Rdb/VMS, for example, is written in two pieces:

- A multithreaded, on-line piece (the client), that handles forms requests from multiple users

- A single-threaded, database server piece (a server instance), that performs the actual synchronous database I/O

This client/server approach

requests are issued, this environment may not be recommended for applications that perform infrequent I/O or that expect very small numbers of concurrent users, such as end-of-day accounting

to database access is functionally very similar to that of ACMS procedure servers and offers similar benefits. Like the ACMS monitor, the DECintact monitor offers system management facilities to

define pools of servers

and to adjust them dynamically at run-time in accordance with load. Similar algorithms are used in both monitors to allocate server instances to client threads and to start up new instances, as necessary. The DECintact server code, like the ACMS procedure server code, can define initialization and termination procedures to perform once-only start-up and shut-down processing. With DECintact transaction semantics, which are layered on DECdtm services, a client can declare a global transaction that the server instance will join. The server instance can also declare its own independent transaction or no transaction. (In terms of the DECdta architecture, this client/server approach implements the functions of a transaction server.) [1] The principal difference between the DECintact and ACMS approach is that DECintact clients and servers use a message-based 3GL communications interface to send and receive work requests. Control in the ACMS monitor resides in the execution controller.

As the ACMS monitor does, the DECintact architecture addresses the problem areas discussed in the On-line Execution section

Resource Use. The DECintact system's multithreaded methodology economizes on VMS resources. Similar to the method used in the ACMS monitor, the system reduces process creations and image activations. A major difference between the ACMS and DECintact architectures is the way the DECintact monitor implements multithreading support. The transparent implementation of threading capabilities means that coding multithreaded applications is no more difficult than coding traditional single-threaded applications. As with any application-level threading scheme, however, the responsibility for ensuring that a logic error in one thread is isolated to that thread lies with the application. The DECintact client/server facilities for accessing databases, like those used in the ACMS monitor, can realize similar benefits in process reuse, throughput, and error isolation.

Start-up Costs. The DECintact architecture, like the ACMS architecture, distributes start-up costs (i.e., system resources and elapsed time) between two points: the start of the DECintact system, and the start of applications. System start-in several ways. Also, as with the ACMS approach, the factors we chose to trade off allowed us to achieve better efficiency, performance, and ease of use.

up can involve prestarting VMS process shells (as discussed previously) for subsequent application image activation. On-line application start-up is executed on demand when the first user selects a particular menu task. Multithreaded applications, once started, do not exit but wait for new user threads as users select the application. Thus, the DECintact terminal user can switch between application images and incur only an inexpensive thread creation.

Contention. As in the ACMS monitor, database accesses in the DECintact client/server environment are channeled through a relatively few, but heavily used, number of processes rather than through a large number of lightly used processes. This reduction decreases lock contention.

Processing Location. Forms processing can be off-loaded to a front end and brought closer to the terminal user. Thus smaller, less expensive CPUs can be used while the rest of the application executes on a larger back-end machine or cluster. In the DECintact monitor, the front end can consist of forms processing only or a mix of forms processing and application remote queuing work.

Off-line Execution

Many transaction processing applications are used with requirements that differ from the requirements of interactive applications: tasks must be simple data entries, and the system must handle failures transparently.

ACMS Off-line Execution

The ACMS monitor's goal for off-line processing is to allow simple transaction capture to continue when the application is not available. A typical example is the continued capture of data on a manufacturing assembly line by a MicroVAX system when the application is unavailable. The ACMS monitor provides two mechanisms for supporting nonterminal devices: queuing agents and user-written agents.

Figure 4 illustrates the ACMS queuing model. A queuing system is a resource manager that processes entries, with priorities, in first-in, first-out (FIFO) order. (In terms of DECdta, this is the queue resource manager.) [1] The ACMS queuing facility is built upon RMS-indexed files. The primary goal of ACMS queuing is to provide a store-and-forward mechanism to allow task requests to be collected for later execution. By using the nonterminal devices, such as a bar code reader or a communications link used for an electronic funds transfer application. Because there is no human

interaction with these applications, they have two

ACMS$ENQUE_TASK service, a user can write a process that captures a task request and safely stores the task on a local disk queue.

The ACMS monitor provides a special agent, called the queued task initiator (QTI), which takes a task entry from the queue and submits it to the appropriate execution controller. The QTI starts a DECdtm transaction, removes the task entry from the queue within that transaction, invokes

the ACMS task, and passes the transaction identifier. (In the DECdta architecture, the QTI implements the functions of a request manager for queued requests.) [1] The task then joins that transaction. The removal from the queue is atomic with the commit of the task, and no task entry is lost or executed twice.

Figure 5 shows the ACMS user-written agent model for off-line processing. With the ACMS system interface, users may write their own versions of the command process. Note that

because these agents cannot be safely stored on disks, this method is generally not as reliable as using queues. User-written agents can be used, however, with DECdtm and the fault-tolerant VAXft 3000 system to produce a reliable front-end system. To do so, a user writes an agent completes, the agent commits the transaction. If DECdtm returns an error on the commit, the agent loops back to start another transaction and to resubmit the task. If a VAXcluster system is used for the application, this configuration will survive any single point of failure.

DECintact Off-line Execution

The DECintact monitor provides several facilities for applications to perform off-line processing. These facilities allow applications to

- Interface with and process data from nonterminal devices and asynchronous events

- Control transaction capture, store and forward, interprocess communication, and business work flow through the DECintact queuing subsystem

Off-line Multithreading. Off-line, multithreaded DECintact applications are typically used to service asynchronous events, such as the arrival of an electronic funds transfer message or the addition to the queue of an item already on a DECintact queue. The that captures the input for the task and then starts a DECdtm transaction. The agent uses the system interface services to invoke the ACMS task and passes the transaction identifier and the input data. When the task call

application programmer explicitly controls how many threads are created, when they are created, and which execution path or paths each thread will follow. Off-line, multithreaded applications are well-suited to message switching systems and other aspects of electronic funds transfer in which each thread may be dedicated to servicing a different kind of event.

DECintact Queues. The primary goal of the DECintact queuing subsystem is to support a work flow model of business transactions. (In the DECdta architecture, the DECintact queuing subsystem implements the functions of a queue resource manager and request initiator for queued requests.) [1] In a typical DECintact application that relies on queuing, the state of the business transaction may be represented by the queue on which a particular queue item resides at the moment. An item moves from queue to queue as the item's processing state changes, much as a work item moves from desk to desk. The superset of queue items that reside on queues throughout the application at any one time represents the state of transactions currently executing. Depending on the number of programs that need to process data during the course of a transaction, a queue item may be inserted on several different queues before the application: from the front end, where queues collect and route incoming data; to the back end, where queues can be integrated with data files in recovery units; and in between, where different programs in the application can use queues to share data.

The DECintact queuing subsystem consists of a comprehensive set of callable services for the creation and manipulation of queues, queue sets, and queue items. Queue item operations performed within the context of a DECintact transaction are fully atomic along with DECintact file operations.

In addition to overall workflow control, the DECintact queuing system allows the following:

- Deferred processing-An item can be queued by one process and then removed from the queue later by another process for processing. Deferred processing is useful when the volume of data entry is concentrated at particular times of day; applications can assign themselves to one or more queues and can be notified when an item is inserted on the queue.
- Store-and-forward processing-When users

transaction completes. The application also may wish to chain together several small transactions within the context of a larger business transaction. The DECintact queuing system functions throughout the

at the front end of the system write items to local queues, data entry can be continuous in the event of back-end system failure or whenever a program that is needed to process data is temporarily unavailable.

o Interprocess communication-Locally between applications sharing a node and by means of the DECintact remote queuing facility, applications can use the queuing system to reliably exchange application data between processes and applications.

A fundamental difference between ACMS queues and DECintact queues is that the ACMS system inserts tasks onto the queues, and the DECintact system inserts data items. In DECintact queuing, each data item contains both user-supplied data and a header that includes an item key and other control information. The header is used by the queuing system to control the movement of the item from queue to queue. Each queue item can be assigned an item priority. Items can be removed from the queue in FIFO order, in FIFO order within item priority, or by direct access using the item key. Queues can be stopped and started for insertion, removal, or both. Queues can also be redirected transparently at the system management level to running applications. items can be held against removal or released. Queues can be grouped together into logical entities, called queue sets, which look and behave to the application the same as individual queues. Queue sets have added facilities for broadcast insertion on all members of a queue set and a choice of removing algorithms that can weight relative item- and queue-level priorities from the queue.

DECintact queues can be automatically distributed. At the system management level, a local queue can be designated as remote outbound. That is to say, items added to this queue are shipped transparently across the network to a corresponding remote inbound queue on the destination node. The transfer is handled by the DECintact queuing system by using exactly-once semantics (i.e., the item is guaranteed to be sent once and only once). From the point of view of the application that is adding or removing items from the queue, remote queues behave exactly as local queues behave.

To better understand some of the uses for DECintact queuing, consider a simplified but

In the DECintact monitor, alert thresholds can be specified on a queue-by-queue basis to alert the system manager when queue levels reach defined amounts. Individual queue

representative electronic funds transfer example built on the DECintact monitor. Figure 6 shows the elements of such an application. In this application, transactions might be initiated either locally by clerks entering data into the system from user-generated documents or by an off-line application that receives data from another branch or bank. The transactions are verified or repaired by other clerks in a different department of the bank. The transactions are then sent to destination banks over one or more network services.

To implement this application, the developer uses queues to route, safely store, and synchronize data as it progresses through the system, and to prioritize data items. Data items are given priority levels, based on application-defined criteria, such as transfer amount, destination bank, or time-to-closing.

As illustrated in Figure 6, the terminal manager controls terminals for the Data Entry and Verify and Repair applications. Clerks enter data from user-generated documents on-line as complete messages. Verification and repair clerks receive these messages as work items from the verify and repair queue through the Verify and Repair application. The result of verification is either a validated message, which is ultimately sent to a destination bank, or an unverifiable message, which is routed to the supervisor queue for special handling. After special handling, the message rejoins the processing flow by returning to the verify and repair queue. After validation, the messages are inserted in the Fedwire Xmt queue and sent over the network means of database server programs.

The Fedwire Xmt queue could be defined as a queue set, which would permit the Fedwire Process application to remove items from the queue by a number of algorithms that bias the transfer amount by queue and item priority. Similarly, this queue set could be passively reprioritized near the close of the business day. In other words, the DECintact system administrator could use the DECintact queue utility near the end of the day to change queue-wide priorities and ensure that items with a higher priority level in the queue set would be sent over the Fedwire first, without changing any application code.

to the Federal Reserve System. The Fedwire Process application controls the physical interface to the communication line and implements the Fedwire protocol. The validated messages are also used to update a local database by

Application Management

Typically, transaction processing applications are crucial to the business running the applications. If the applications cannot perform their functions reliably or securely, business activity may have to cease altogether or be curtailed, as in the case of an inventory control application or electronic funds processing application. Therefore, the applications require additional controls to ensure that the applications and the access by users to the applications are limited to exactly what is needed for the business.

ACMS Application Management

Of the many features and tools for monitoring and controlling the system offered in the ACMS monitor, three areas are most often used.

- o Controlling and restricting terminal user environments
- o Controlling and restricting the application
- o Ability to dynamically make changes to the application without stopping work

In addition to using the VMS user authorization file (VMS SYSUAF), the ACMS monitor provides utilities to define which users and terminals have access to the ACMS system. Controlled terminals are terminals defined by one of these utilities to be owned by user has login access, the VMS system cannot be accessed. The user's access is restricted to only those ACMS functions that the user is permitted to invoke. This restriction prevents a user from damaging the integrity of data on the system. The ACMS monitor also allows access support for terminals that are automatically logged in to the ACMS system, such as a terminal on a shop floor. Such access is useful for unprivileged users who are not accustomed to computers. They can enter data without understanding the process for logging in to the system.

For application control, the ACMS monitor uses a protected directory, ACMS$DIRECTORY, to store the application definition files. The application authorization utility (AAU) ensures that special authorization is required for a user to make changes to an application.

In the ACMS monitor, the application is a single point of control. The ACMS /START APPLICATION and ACMS/STOP APPLICATION commands cause the execution controller for the application to be created and deleted. An operator can control the

the ACMS monitor. These terminals are allocated by the ACMS monitor when the ACMS system is started. When a user presses the Return key, the ACMS monitor displays its login prompt. Unless the

times when an application is accessible. For example, an application can be controlled to run only on Fridays or only between certain hours. The control of access times can also be used to restrict access while changes or repairs are made to the application. This type of access control is difficult to achieve with only the VMS system because the VMS system does not provide these capabilities.

The execution controller does access-control list checking that is specified for each task. This mechanism can restrict user access by function. For example, a user could have the privilege to make a particular update to a database but not have access to read or make changes to any other parts of that database. The execution controller achieves a much finer level of control than do the mechanisms of the VMS system or the database system.

DECintact Application Management

The DECintact monitor controls access to the whole system and to individual tasks by means of a security subsystem. The subsystem adds transaction-processing-specific features to basic VMS security.

o User security profiles specify the DECintact given DECintact user name can be signed on to the DECintact system at any one time on any one node.
o Dedicated terminal security profiles are used, in conjunction with user security profiles, to provide geographic entitlement.
o CAPTIVE and INITIAL_MENU user attributes restrict users to a specific menu level of functions and prevent the user from accessing outer levels.
o User-specific menus are menu entries for which an explicit authorization has been granted in the user profile and are the only menu items visible on the menu presented to terminal users. The DECintact monitor does include an exception for users who have an auditor privilege. Auditors can see all menu functions but must be specifically authorized to execute any single function.
o The subsystem provides the ability to dynamically enable or disable specific menu functions.

user name and password (DECintact users are not required to have an entry in the VMS SYSUAF file); levels of security entitlement; inclusive and exclusive hours of permissible sign-on; menu entries

authorized for the user. Only one user under a

- Password revalidation is an attribute that can be associated with a menu function. If set, the user must reenter the DECintact user name and password before being allowed to access the function.

The DECintact monitor supports both controlled or dedicated terminals and terminals assigned LAT terminal server application ports, as does the ACMS monitor. These terminals are owned by, and allocated to, the DECintact system. When a user types any character at these terminals, a DECintact sign-on screen is displayed, and the user is prevented from logging in to the VMS system.

Geographic entitlement limits certain DECintact terminal-based functions to certain terminals or even to certain users on certain terminals. The three elements in geographic entitlement are as follows:

- o The user security profile enables a function to be accessed by a certain user.
- o The terminal security profile enables a function to be accessed at a certain terminal.
- o A GEOG attribute is associated with a menu entry in the terminal manager/dispatcher's menu database. This attribute, when associated with a function, demands that there be an applicable terminal security profile before

Normally, if a function is enabled in a user profile, the user can access the function without further checks. If the GEOG attribute is associated with the function, however, that function must be enabled in the user profile and in the terminal profile before it can be accessed. Geographic entitlement is frequently a requirement in financial environments which have specific and rigid security protocols.

For example, a bank officer may be authorized to execute certain sensitive functions available only at dedicated terminals when the officer is signed-in at the home office. The same officer may be authorized to execute only a subset of less sensitive functions when signed-in from a branch office. Such sensitive functions can be protected by requiring that the user profile and the dedicated terminal profile enable the function.

Applications and resources are controlled within the context of a DECintact copy's run-time and management environment. Multiple copies can be established on the same VMS system. Different groups of users can maintain a certain level of autonomy (e.g., separate the function can be accessed.

applications and data files), but all users can

also share some or all functions and resources of a given DECintact version. A typical example of this concept, that is, the ability to create multiple DECintact copies for isolation and partitioning, is the common practice of establishing development, acceptance testing, and production DECintact environments. Managing applications and resources within a development environment, for example, can differ from managing applications and resources within a production environment with a different system manager.

Access to menu functions is controlled by the INTACT MANAGE DISABLE /ENABLE command. This command removes or restores specified functions dynamically from all menus in the DECintact copy and disables or enables their selection by subsequent users. (Current accessors of the specified function are allowed to complete the function.) The execution of single- and multithreaded applications or DECintact system components can be shut down by the INTACT MANAGE SHUTDOWN command. This command issues a mailbox request to the application or component, which then initiates an orderly shutdown. Access to the system by inclusive and exclusive time of day is controlled on a per-user basis through the DECintact security subsystem.

queue set attributes, and performs all other functions necessary for managing the DECintact queuing subsystem.

In general, the DECintact monitor's security and application control focuses on the front end by concentrating access checking at the point of system sign-in and menu generation. The ACMS system concentrates more on the back-end parts of the system by means of VMS access control lists (ACL) on specified tasks. The ACMS approach is built on VMS security and system access (the SYSUAF file) and reflects an environment in which the VMS system and the transaction processing security functions are typically performed by the same system management agency. The DECintact monitor's system access is handled more independently of the VMS system and reflects an environment in which transaction-processing-specific security functions may be performed by a different department from those of the general VMS security system.

Conclusion

The ACMS and DECintact transaction processing

In addition to these commands and functions, the queuing subsystem is managed by means of a queue management utility. This utility creates and deletes queues and queue sets, modifies queue and

monitors provide a unified set of transaction-processing-specific services to the application environment. A large functional overlap exists between the services each monitor provides. Where the functions provided by each monitor are identical or similar (e.g., client /server database access and support for DECforms), the factors that distinguish one from the other are primarily a result of the use of 4GL and 3GL application programming styles and interfaces. Where notable functional differences remain (as in each product's respective queuing or security systems), the differences are primarily ones of emphasis rather than functional incompatibility. The set of common features shared by both monitors has been growing with the latest releases of the ACMS and DECintact monitors. This external convergence has been fostered and made possible by an internal convergence, which is based on sharing the underlying code that supports the common features of each monitor. As more common features are introduced and enhanced in the DECtp system, the investment in applications built on either monitor can be protected and the distinctive programming styles of both can be preserved.

Reference

1. Philip A. Bernstein, William T. Emberton, and Vijay Trehan, "DECdta-Digital's Distributed Transaction Processing Architecture," Digital Technical Journal, vol. 3, no. 1 (Winter 1991, this issue): 10-17.