The Structure of the OpenVMS Management Station

by James E. Johnson

ABSTRACT

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

OVERVIEW

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

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

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

Data from customer surveys indicated the need for a quick response to the problems of managing OpenVMS systems. For this reason, the project team chose a phased delivery approach, in

which a series of frequent releases would be shipped, with support for a small number of system management tasks provided in an individual release.

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

MANAGING OPENVMS USER ACCOUNTS

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

For example, Table 1 gives the attributes of an OpenVMS user stored in various files, including the user authorization file (SYSUAF.DAT), the rightslist file (RIGHTSLIST.DAT), and the DECnet network proxy file (NET$PROXY.DAT). Prior to the OpenVMS Management Station product, these files were managed by a collection of low-level utilities, such as AUTHORIZE. Although these utilities provide the ability to manipulate the individual user attributes, they offer little support for ensuring that the overall collection of user attributes is consistent. For instance, none of these utilities would detect that a user's account had been created with the user's home directory located on a disk to which the user had no access.

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

Data Store	Attributes	Management Utility
SYSUAF.DAT	Username, Authorization data (e.g., passwords), process quotas, login device, and directory	AUTHORIZE
RIGHTSLIST.DAT	Rights identifiers	AUTHORIZE
NET$PROXY.DAT	Remote<->local user	AUTHORIZE

	DECnet proxy mappings	
VMS$MAIL_PROFILE.DAT	User's mail profile	MAIL
QUOTA.SYS (per disk)	User's disk quota	DISKQUOTA
Login directory	User's home directory	CREATE/DIRECTORY
TNT$UADB.DAT	User's location, phone number, and organization information	<new with OpenVMS Management Station software>

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

OPENVMS SYSTEM CONFIGURATIONS

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

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

[Figure 1 (Distributed OpenVMS System Configuration) is not available in ASCII format.]

Table 2 Usage and User Community for Sample Configuration

Name	Usage	User Community

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

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

Supporting Parallel Execution

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

Leveraging VMScluster Attributes

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

In the first case, the scope of the resource extends throughout the VMScluster system. Here, it is desirable (and when the operation is not idempotent, it is necessary) for the operation

to execute once within the VMScluster system. In the latter case, the operation must execute against every system within the cluster that the system manager wants to affect. Also, the set of resources that falls into the first case or the second is not fixed. In the OpenVMS operating system releases, the ongoing trend is to share resources that were node-specific throughout a VMScluster system. The OpenVMS Management Station software must handle resources that have different scopes on different systems that it is managing at the same time.

Wide Area Support

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

OPENVMS MANAGEMENT STATION STRUCTURE

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

[Figure 2 (OpenVMS Management Station Structure) is not available in ASCII format.]

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

The server software on the OpenVMS systems consists of a message-dispatching mechanism and a collection of server OMMs that enact the various management requests. The dispatcher is also responsible for forwarding the management request to all target VMScluster systems and independent systems, and for gathering the responses and returning them to the client. The version 1.0 server contains two OMMs; UAServer and Spook. The former implements the server support for both the OpenVMS Accounts and OpenVMS User OMMs. The Spook OMM implements the server component of the authentication protocol.

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

Design Alternatives

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

Communications Layer Placement.

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

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

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

systems that it may ever manage.

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

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

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

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

On a purely pragmatic level, an SNMP engine was not present on the OpenVMS operating system. The CMIP-based extensible agent that was available exceeded the management software's limits for resource consumption and responsiveness. For instance, with responsiveness, a simple operation using AUTHORIZE, such as "show account attributes," typically takes a second to list the first user account and is then limited by display bandwidth. For

successful adoption by system managers, the project team felt that any operation needed to be close to that level of responsiveness. Early tests using the CMIP-based common agent showed response times for equivalent operations varied from 10 to 30 seconds before the first user was displayed. Remaining user accounts were also displayed more slowly, but not as noticeably.

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

CLIENT COMPONENT

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

Grouping OpenVMS Systems for Management Operations

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

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

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

[Figure 3 (Management Domain View) is not available in ASCII

format.]

Participation in the Authentication Protocol

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

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

The hashed password is stored in memory at the client and used for two further purposes. First, if the server connection fails, the client attempts to silently reconnect at the next request (if a request is outstanding when the failure occurs, that request reports a failure). This reconnection attempt also undergoes the same authentication exchange. If the hashed password is still valid, however, the reconnection is made without apparent interruption or requests for input from the system manager. Second, the hashed password is used as a key to encrypt particularly sensitive data, such as new passwords for user accounts, prior to their transmission to the server.

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

Display and Update of User Account Information

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

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

For instance, all the mail profile information is in one window.

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

[Figure 4 (User Characteristics Display) is not available in ASCII format.]

The OpenVMS User OMM and the zoom display organize the attributes into logical groupings, simplify the display and modification of those attributes, and provide fairly basic attribute consistency enforcement. The project team did encounter one case in which no standard text display proved sufficiently usable. This was in the area of access time restrictions. All attempts to list the access times as text proved too confusing during usability testing. As a result, the project developers produced a specialized screen control that displayed the time range directly, as shown in the Time Restrictions section of Figure 5. Later, system managers who participated in the usability testing found this to be very usable.

[Figure 5 (User Time Restrictions Display) is not available in ASCII format.]

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

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

For example, one interesting case involves user account resource quotas. One reason a system manager changes these settings is to accommodate a new version of an application that needs increased values to function correctly. Prior to the development of the OpenVMS Management Station tool, the system manager had to locate all the users of this application, examine each account, and increase the resource quotas if they were below the application's

needs. Conversely, with the OpenVMS Management Station product, the system manager selects the users of the application in the domain display (Figure 3), and requests the zoom display for the entire set. The system manager then proceeds to the user quota display and selects the quotas to change. The selection takes the form of a conditional request--in this case an At Least condition--and the value to set. The system manager then presses the Apply to All button, and the changes are carried out for all selected users. Figure 6 shows the user quota display.

[Figure 6 (User Quota Display) is not available in ASCII format.]

COMMUNICATIONS COMPONENT

The communications component is responsible for managing communications between the client and servers. It provides support for transport-independent, request-response communications, automated reconnection on failure, and support routines for formatting and decoding attributes in messages.

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

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

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

In the end, the OpenVMS Management Station software contained a communications layer that hid many of the details of the underlying implementation, offering a simple request-response paradigm. The only difference with an RPC-style model is that the data encoding and decoding operations were moved into support routines called directly by the sender or receiver, rather than by the communications layer itself. In future versions, the goal for this layer is to support additional transports, such as

simple Transmission Control Protocol/Internet Protocol (TCP/IP) messages or DCE's RPC. An investigation into providing additional transports is currently underway.

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

Client Request-response Mechanisms

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

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

Reconnection on Failure

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

If an existing communications link fails, that link is marked as unconnected. If a response were outstanding, an error would be returned instead of a response message. When the communications layer is next called to send a request across the unconnected link, an automated reconnection is attempted. This involves establishing a network connection to a target system in the request. Once the connection has been established, the authentication protocol is executed, using the previously

supplied authentication data. If authentication succeeds, the request is sent. If it is a continuation request, and the target server has no existing state for that request, an error response is returned.

At most, the resulting behavior for the system manager is to return an error on a management request, indicating that communication was lost during that request's execution. If no request was in progress, then there is no apparent disruption of service.

Message Encoding and Decoding

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

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

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

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

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

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

Message encoding is supported by a set of routines. The first accepts an attribute code and its associated data item. It appends the appropriate message item at the end of the current

message. This is used to encode both requests and responses. The second routine takes a message buffer and an attribute code, returning the attribute's value and a status code indicating if the attribute was present in the message buffer. The client uses this routine to locate data in a response. The third routine takes a message buffer, a table listing the attribute codes that are of interest to the caller, and an action routine that is called for each message item that has an attribute code found in the table. The server OMMs use this routine to process incoming requests.

Handling of Complex Data Types

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

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

SERVER COMPONENT

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

Server Infrastructure

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

[Figure 7 (Server Infrastructue and Message Flow) is not available in ASCII format.]

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

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

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

Server OMM Structure

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

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

allows normal access checking and auditing in the OpenVMS operating system to work transparently.

[Figure 8 (UAServer OMM) is not available in ASCII format.]

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

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

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

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

SUMMARY

The OpenVMS Management Station tool has demonstrated a robust client-server solution to the management of user accounts for the OpenVMS operating system. It provides increases in functionality

and data consistency over system management tools previously available on the OpenVMS operating system. In addition, the OpenVMS Management Station software is focused on the management of several loosely associated VMScluster systems and independent systems. It has addressed the issues concerning performance, usability, and functionality that arose from the need to issue management requests to execute on several target systems.

ACKNOWLEDGMENTS

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

REFERENCES

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

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

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

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

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

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

BIOGRAPHY

James E. Johnson A consulting software engineer, Jim Johnson has worked in the OpenVMS Engineering Group since joining Digital in 1984. He is currently a member of the OpenVMS Engineering team in Scotland, where he is a technical consultant for transaction processing and file services. His work has spanned several areas

across OpenVMS, including RMS, the DECdtm transaction services, the port of OpenVMS to the Alpha architecture, and OpenVMS system management. Jim has retained an active interest in transaction processing, especially the area of commit protocols. Jim holds one patent on commit protocol optimizations. He is a member of the ACM.

TRADEMARKS