Inheritance Based Environments in Stand-alone OpenVMS Systems and OpenVMS Clusters

By Robert Gezelter, CDP, CSA, CSE
Software Consultant
35-20 167th Street, Suite 215
Flushing, New York 11358-1731 USA
+1 (718) 463 1079
http://www.rlgsc.com
E-mail: Gezelter@rlgsc.com

Abstract

The goal of stand-alone or clustered configurations is to present the user, and application, with a consistent operating environment regardless of the physical technologies actually used to implement that environment.

Many OpenVMS Cluster systems include a multitude of different processors, each with different peripheral configurations and capabilities. Many organizations also have multiple stand-alone OpenVMS systems running identical or nearly identical applications on different hardware configurations. Disaster tolerant (DT) configurations can further complicate the environment by introducing propagation delay and other consequences of physical differences into the environment.

OpenVMS clustering technology was introduced before the popularization of object-oriented terminology and the codification of its concepts. However, object-oriented concepts, particularly inheritance, well describe OpenVMS in clustered environments. Principles of inheritance are particularly apt when implementing functionally identical environments upon different physical environments. Architectures employing inheritance realize significant reductions in total cost of ownership (TCO) and correspondingly large improvements in portability and operational transparency.

In contrast with many other operating systems, the iterative nature of the OpenVMS logical name facility enables the use of multiple levels of translation with corresponding default values for each level. This flexibility permits OpenVMS systems to assimilate dramatic changes in operating environment with a change to a single logical name at a variety of levels, dramatically reducing TCO.

Introduction

The standard OpenVMS user environment conceptually rests upon two related, yet distinct logical hierarchies:

- the four hierarchical levels of the logical name environment[1] and
- the execution of the system-wide and user-specific login profiles[2]

The classification of users into groups, the separate identification of privileged user groups (those identified as "System"), and all other users, suggests a natural hierarchical structure. While these hierarchies are more than what many other operating systems provide, they still are not fully reflective of many environments. The organization, the applications, and the computing environment of today's corporate organization are often more complex than a single organization with easily identifiable, disjoint groups.

Many user environments are far richer in diversity than they appear at first glance, reflecting users' different collections of applications, their roles, and their responsibilities.

Computing environments are more than single instances of processors, memories, and peripherals. The most critical elements of a well-configured OpenVMS environment reside in the logical environment created by the system manager and application architect, not in the specification of the bare hardware and software environment. The exact processor (VAX, Alpha, or Itanium) and peripheral configuration of the system is far less important.

User environments are hierarchically nested. For example, an individual user is typically a member of a group. In its turn, the group is a part of a company. Thus, an individual's application environment depends upon that person's place in the organization. Similarly, the group or department's environment is merely an instance of the standard company-level environment, tailored to the specific functions performed by that department. In a service bureau or Applications Service Provider (generally referred to as an "ASP") environment, where multiple companies (or several sibling companies) share a system, there are also company-wide environments, which are shared by all users at an individual company but differ to some extent between different companies.

---

[1] **LNM$SYSTEM** (composed of **LNM$SYSTEM_TABLE** and **LNM$SYSCLUSTER**, which in turn really translates to **LNM$SYSCLUSTER_TABLE**), **LNM$GROUP**, **LNM$JOB**, and **LNM$PROCESS** are part of the user's context created as part of the processing performed by **$CREPRC**.

[2] The system-wide login command file is located in **SYS$MANAGER:SYLOGIN.COM**; by default, the user-specific login is in the user's default directory. The user's login can alternatively be placed in any private or shared file that is accessible to the user through setting the appropriate fields of the user's record in the system UAF.

A review of the mechanics of what happens when a user connects to an OpenVMS system is appropriate at this point.

When a user logs on to an OpenVMS system, **LOGINOUT.EXE** creates a basic operating environment consisting of:[3]

- the default directory, **SYS$LOGIN**
- the device upon which the default directory is located, **SYS$LOGIN_DEVICE**
- a scratch device/directory, **SYS$SCRATCH**
- the default device characteristics established by the command procedures executed as part of the login process
- the logical name environment, which is a hierarchical list of names and translations of names. The logical name tables are searched in a variety of situations when commands and programs access a variety of resources, most commonly files and queues. Generally speaking, translation continues until no more translations are possible, each iteration starting again from the beginning. This feature is dramatically different from the one-pass symbolic parameters available on other systems. A user process's actual logical name context is built by **LOGINOUT.EXE** and includes:
    - the command files executed as part of login processing
    - the job logical name table, specific to the job, referred to by the name **LNM$JOB**[4]
    - the group logical name table, applicable to all users who share the same UIC Group number, **LNM$GROUP**[5]
    - the system logical name table, **LNM$SYSTEM**

---

[3] For simplicity, we refer to the basic case of an interactive user. The start of a network or batch produces similar results and follows a similar path. Creating processes that are neither interactive, batch, nor network may create a similar environment, or may result in a slightly truncated environment (e.g., whether the **/AUTHORIZE** option is used on the **RUN/DETACH** command). Environmental truncation has important implications for the proper operation of applications and facilities.

[4] **LNM$JOB** is itself a logical name, whose translation resides in the **LNM$PROCESS_DIRECTORY** as **LNM$JOB_XXXXXXXX**, where XXXXXXXX is the eight-digit hexadecimal address of the Job Information Block (see Goldenberg, Kenah, Dumas, "VAX/VMS Internals and Data Structures, Chapter 35, page 1073, first footnote)

[5] **LNM$GROUP** is itself a logical name, whose translation resides in the **LNM$PROCESS_DIRECTORY** as **LNM$GROUP_GGGGGG**, where GGGGGG is the six-digit octal UIC group number (ibid.)

    - the cluster logical name table, **LNM$CLUSTER_WIDE**[6]

- other process characteristics established as a part or consequence of login processing.

These capabilities are generic, and straightforwardly support the construction of environments based upon a simple cluster belonging to a single entity with groups of users, each with their own profiles. However, environments are often not as straightforward. Traditionally, complex environments have been implemented by explicitly invoking the particulars of each environment from the individual user's **LOGIN.COM** file. A user's specific environment is achieved by manually adding elements to the user's login profile. This method requires inordinate maintenance, is excessively fragile, and is virtually impossible to manage.

[6] Introduced in OpenVMS Version 7.2 (OpenVMS Version 7.2 New Features Manual, Order#AA-QSBFC-TE, July 1999)

Philosophic Basis

OpenVMS provides a straightforward approach, founded upon the same architectural principles as the operating system itself, which dramatically simplifies the creation of customized environments. This can be accomplished with a minimum of complexity and maintenance effort, and a high degree of manageability and scalability.

The basis of the simplification is the realization that while environments differ dramatically, they do so in an orderly, systematic way, and those differences can be implemented with minor, manager-level changes.

An approach based upon axes of variation, leveraging the strengths of OpenVMS and its hierarchical logical name structure, is more robust, more auditable, and more maintainable than the traditional explicit enumeration approach.

Environments can be characterized by multiple, independent axes of variation. For the purposes of this paper, we will consider five axes of variation, although the concept can be easily extended to address further axes of variation.

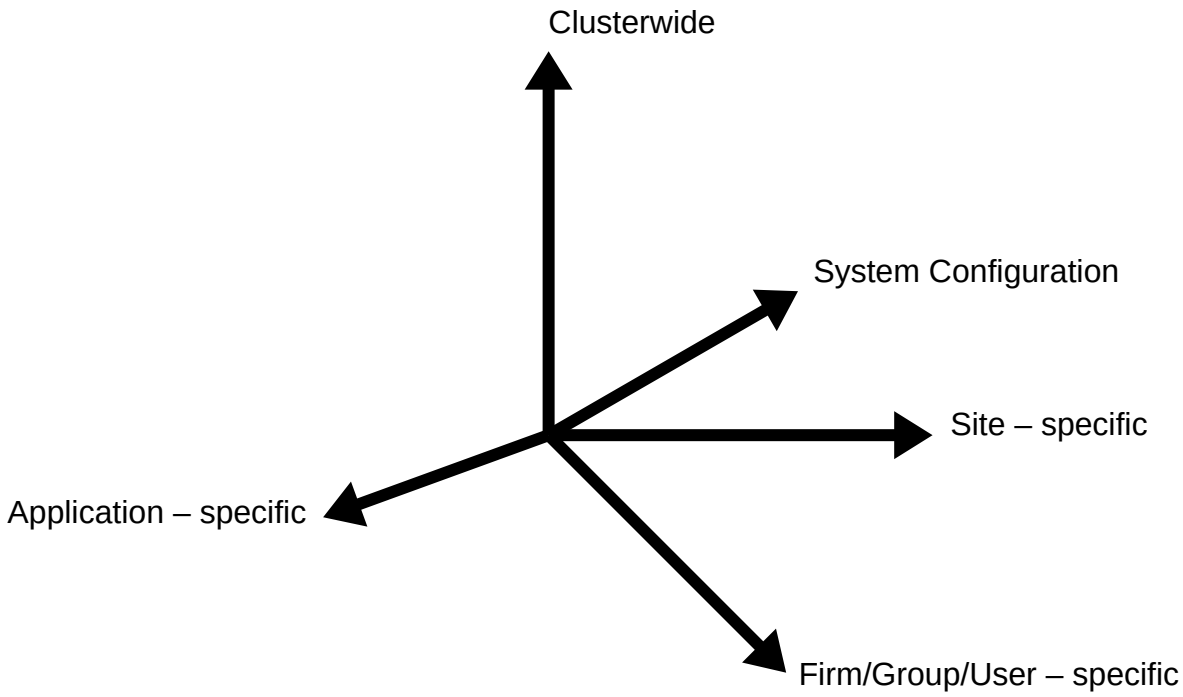

Figure 1 – Five illustrative independent axes of variation

The axes are considered independent; changes in the names comprising a single axis do not imply changes in different axes. There is also no need to restrict name translation iterations to a single axis.

There are five axes of variation referred to in this paper, namely:

Clusterwide Variations

Clusterwide variations reflect those differences that define the individual cluster as a whole.

Clusterwide variances are best dealt with using the clusterwide logical name table.[7] The clusterwide local name table is automatically included in the logical name resolution path, **LNM$FILE_DEV**, at a lower precedence than that of the system-wide logical names contained in **LNM$SYSTEM**.

System Configuration Variations

System specific variations reflect the particular needs of a particular system and its hardware.

System specific variations are best dealt with by adding appropriate logical name definitions to the standard system logical name table, **LNM$SYSTEM**.

Site-Specific Variations

Site-specific variability reflects the connection and capability differences that exist on a specific site that is a member of a multi-site OpenVMS Cluster system.

While this axis of variation is not explicitly supported by OpenVMS, it can be added to OpenVMS by the addition of a new, system-wide logical name table inserted in the search path (**LNM$FILE_DEV** in **LNM$SYSTEM_DIRECTORY** or **LNM$PROCESS_DIRECTORY**) between the systemwide and clusterwide logical name tables.

Firm/Group/User-Specific Variations

Some variations are specific to an individual's organization, or place within the organizational hierarchy.[8] OpenVMS traditionally recognizes the group/department and user hierarchy with UIC-based protection and the existence of the group logical name table. This existing axis of variation can be extended and enhanced through slight adjustments to the login processing and the logical name search path.

---

[7] Ibid

[8] Within this paper, we refer to this as firm, group, and user. Additional levels (e.g., division or region) can easily be accommodated by similar means.

Application-Specific Variations

Lastly, some environmental parameters are specific to a particular application. They may overmap similar parameters from similar applications, but differ in that each user of the application must have definitions in his or her logical name search path providing the value for these parameters. Thus, applications can be completely parameter-driven by their environment, with all of the implications of hierarchical defaulting.

One Parameter, One Line of Code

Another overall principal is that each definition should appear, like a common subroutine, only once. Duplicate definitions are a major element in maintenance complexity and costs, as well as an ongoing source of errors.

Dependency

The inclusion of all the definitions in the logical name search path allows different logical names to be phrased in terms of other logical names in the search path. As an example, the location of a file may be expressed as a logical name definition:

```
$ ASSIGN/PROCESS SYS$SCRATCH:VEHICLES.DAT DATABASE
```

which itself includes a reference to a definition in the user's process logical name context (to the logical name **SYS$SCRATCH**). In turn, **SYS$SCRATCH** may include a reference to the logical name **DISK$SCRATCH** which might be defined in the group's logical name table. **DISK$SCRATCH** could also be defined in the system logical name table, **LNM$SYSTEM**.

This is the same approach used by OpenVMS itself. Many logical names are explicitly or implicitly dependent on the definition of **SYS$SYSROOT**[9] or **SYS$SYSDEVICE**. In practice, multiple dependencies of logical names incur a generally insignificant cost.[10]

[9] Most of the **SYS$** logical names are expressed in just such a way, in terms of **SYS$SYSROOT**. For example, **SYS$SYSTEM** is defined as **SYS$SYSROOT:[SYSEXE]**. **SYS$MANAGER** is similarly defined as **SYS$SYSROOT:[SYSMGR]**. **SYS$SYSROOT** is defined as a search list including both the system-specific and clusterwide OpenVMS systems directories. Expanding this scheme to include additional levels beyond system-specific and clusterwide is straightforward.

[10] The processing costs associated with the rapid opening and closing of files and other operations is far more significant. In any event, the labor costs and inflexibility of the approaches required to save the logical name translations are far greater. In applications where thousands of records are processed, the cost of a few extra logical name translations is negligible.

Migrating logical names from private copies in **LNM$PROCESS** to **LNM$JOB** or other logical name tables with a wider purview increases performance by reducing the need to copy large numbers of process-private logical names during **SPAWN** operations.

Inheritance

Successive dependence, illustrated in the previous section, is a powerful technique. Defaulting, a concept implicitly familiar to OpenVMS users in the guise of default file types, and the traditional hierarchy of logical name tables, is a far more powerful mechanism than generally realized. Viewed as a form of inheritance, defaulting is also a mechanism for expressing the variability of user environments.

It has previously been remarked that each axis of variability embodies a hierarchical series of qualifications within the axis. On the individual axis, users are members of groups; groups are members of firms. On the cluster axis, the highest level is the cluster as a single entity; within the cluster, there are sites; within the sites, there are individual cluster nodes.[11]

Hiding the Physical and the Organizational

The purpose of isolating the physical and organizational aspects of a user's environment is the same as the more familiar OpenVMS concepts of disk space management (virtual blocks) and memory management (virtual memory). In the case of disk space and memory management, the purpose is to free the application from managing the details of a specific processor or device environment.

Adapting the same philosophy to users' logical environments similarly allows the re-organization of users and their hardware platforms without the need to explicitly re-engineer each and every reference to the environment.

[11] One could also argue that within nodes, there are individual realizations of nodes on particular hardware. For example, the node **ALPHA** could run at different times on either an ES40 or with a pre-configured alternative configuration for an Alphaserver 1200 as a backup. It is straightforward to implement such an environment, but does not affect the overall discussion.

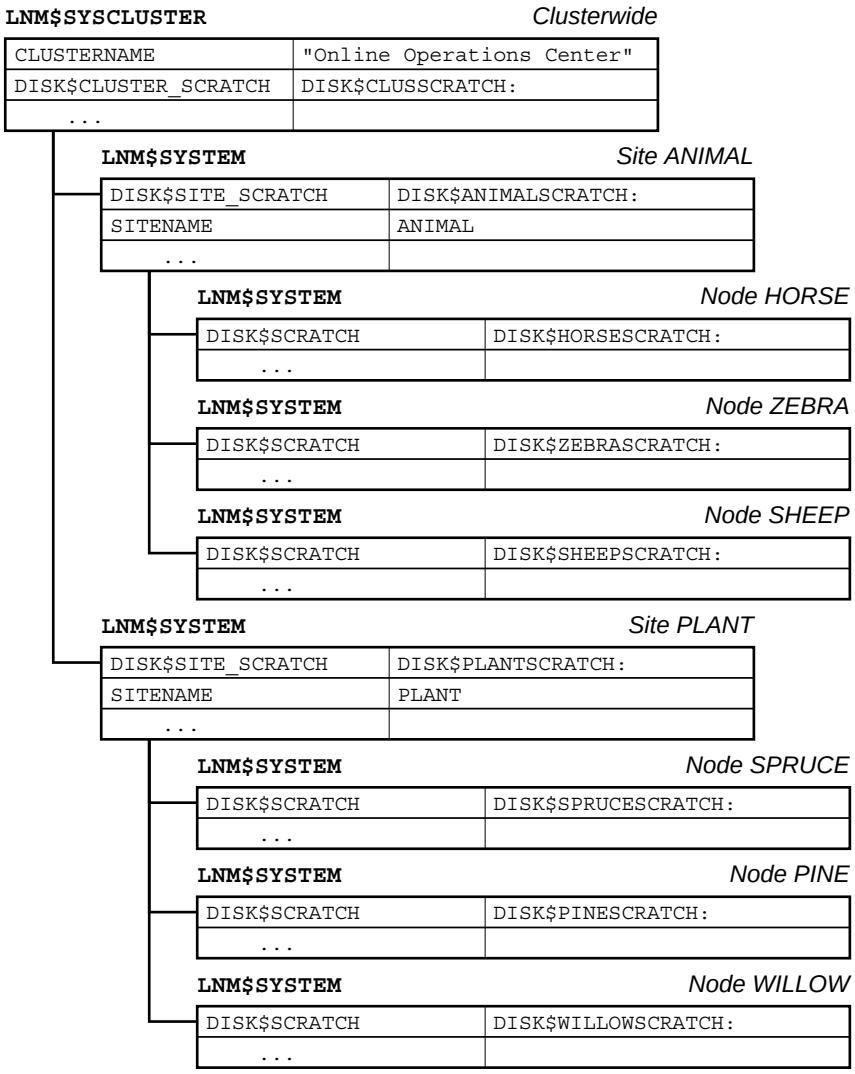

Figure 2 – Hierarchical dependencies and inheritance – Cluster/Site/System

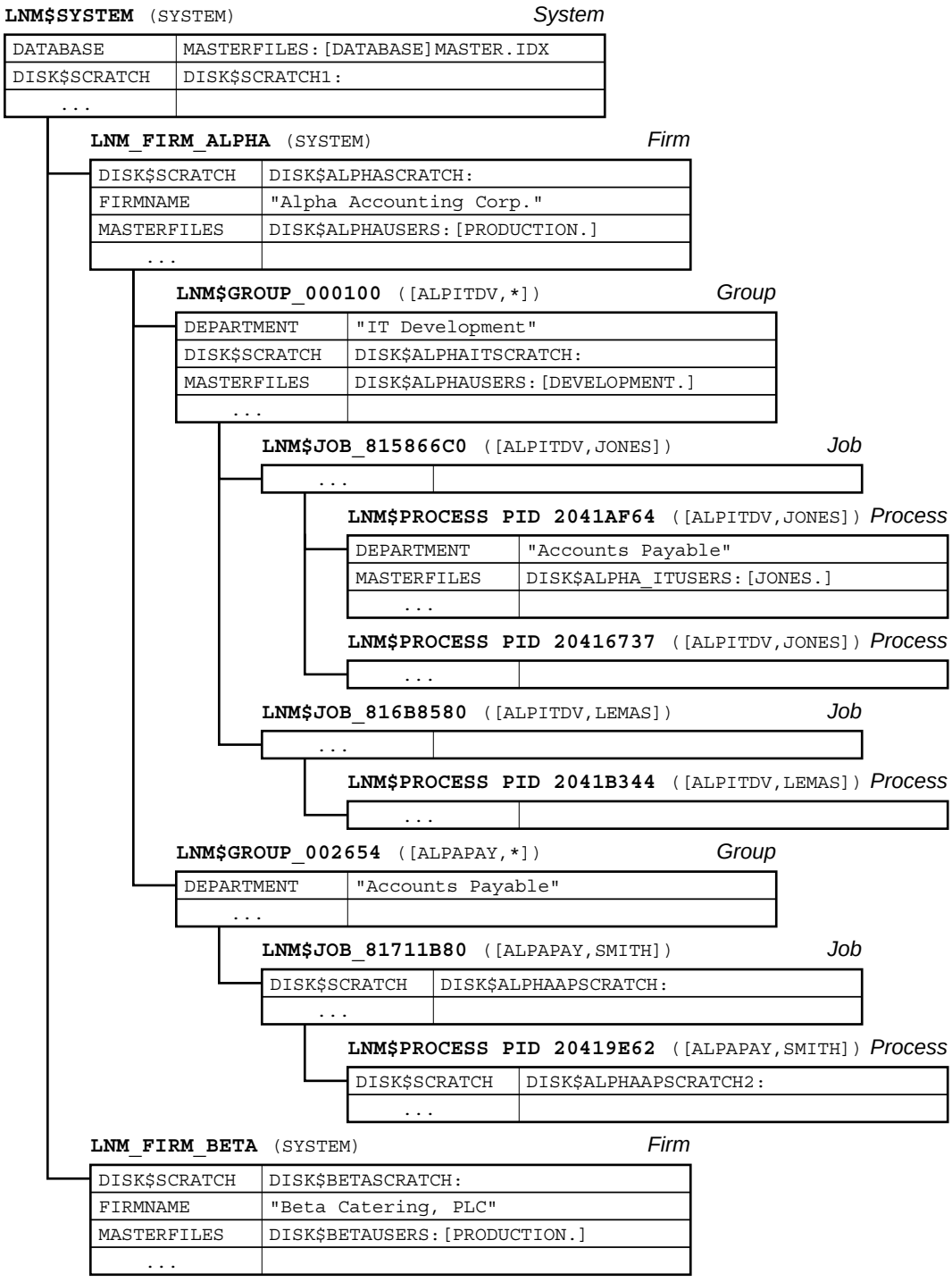

Figure 3 – Hierarchical Dependencies and Inheritance – Firm/Group/User

Physical Configurations

Simple Configuration

Simple systems are employed for a variety of reasons, including cost and space. A large, complex system may not be justified or economically feasible for small organizations or for small applications. A small system may also be used in a large organization or for a large application to support development or to prototype applications.

Small systems, such as a DS10 or similar small workstation or server, are frequently used for testing installation, startup, shutdown, and restart procedures. These operations are extremely disruptive to a large production environment.

The affordability of relatively inexpensive, small OpenVMS platforms permits developers and maintainers to perform these highly disruptive tests with minimal impact on production systems and with a high degree of certainty that the full-scale tests will be successful.

Small systems are also used for projects in the proof-of-concept stage, where economics can make the difference between feasibility and infeasibility.

Advanced Configuration

Larger configurations present more options than small systems. While small systems may be restricted to one or more directly attached disks, a larger configuration may include a mix of directly attached disks (directly attached to the integral SCSI adapters on systems such as the GS-series and ES-series), local disks (CI or SAN attached), and remote SAN or network-attached storage. Each of these storage categories has different operational and performance characteristics.

Different user groups may be assigned to login environments with different attributes, depending upon a multitude of factors. Some factors will be technical in nature (such as a need for large scratch areas, or a need for shadowed and/or mirrored storage) and some will be organizational or political in nature (one department may have contributed the funds for a particular storage facility). In either event, the environment for one group of users (or in some environments, a particular user) will determine the need for that group’s default environment to differ from some other group’s default environment.

Applications Environments

User Disks

During login processing, **LOGINOUT.EXE** determines a user's default disk from the contents of the user's login profile, located in the user authentication file (referred to as the UAF). **LOGINOUT.EXE** populates **SYS$LOGIN** and **SYS$LOGIN_DEVICE** in the user's job logical name table as executive mode logical names (making them available to and usable by privileged images).

Scratch Space

Similarly, **LOGINOUT.EXE** uses the same information to populate the contents of **SYS$SCRATCH**.

Access to Data

When creating a user's process, **LOGINOUT.EXE** also attaches a series of rights list identifiers to the process. These identifiers come from two sources: a set of identifiers that reflect the origin of the process (e.g., **BATCH**, **INTERACTIVE**, **REMOTE**) and those identifiers associated with the user's UAF entry in **RIGHTSLIST**.[12]

[12] The rights list identifiers associated with a process are used to determine access rights to system resources beyond those granted through the normal System/Owner/Group/World and privilege access mechanisms. The details of the access checking scheme and how identifiers are used is described the "OpenVMS Guide to System Security".

Default Inheritance

While not often appreciated, the OpenVMS login processing provides an easy and natural idiom for using at least three levels of inheritance, clusterwide, system, and group.

Concealed, rooted logical names,[13] coupled with the conventions for directory names in FILES-11, provide a natural naming convention and structure for inheritance, as illustrated in Figure 4.

Architectural Concepts

Architects understand that building designs must deal correctly with the forces of nature. A realizable design must always address the realities of construction, the details of the physical materials, and the techniques of fabrication. Designs do not exist in a vacuum, devoid of context.

Systems architecture is a hybrid discipline. In some sense, computer software is totally malleable, purely a creation of the mind of its creator. However, some architectural principles do apply. One such principle is that the overall architecture will only work if the details are correct, both architecturally and in the resulting implementation. For example, Digital's RT-11 (with the "`DK:`" device) and Microsoft Windows™ (with the "`C:`" device) have been hobbled when compared to a system, such as OpenVMS, that hides the identity of the system device behind a logical name.

The principles that apply to the quality of pictures when enlarged or reduced have their analog in the world of OpenVMS configurations. Reducing the size of a picture makes detail imperceptible, but it is still there. When enlarging a picture, detail cannot be invented.

The logical environment used for a larger, clustered system, can be easily reconfigured to transparently provide access to the same resources in a smaller workstation or development system (in effect, the analog to reducing a picture – no loss of picture quality occurs). However, an environment designed for a small system is frequently not suitable when used with a larger configuration (the analog of enlarging a picture – the picture loses sharpness and clarity as it is enlarged). The inherent problem is that the environment designed for the small system does not address the issues that occur in larger configurations.

To an even greater degree, different members of a cluster, whether a conventional OpenVMS Cluster system or a distributed, multi-site disaster tolerant OpenVMS Cluster system, offer the same challenge along a different axis.

---

[13] Much has been written on the use of concealed, rooted logical names, including this author's articles on OpenVMS.org, accessible directly or though the author's www site at http://www.rlgsc.com/publications.html.

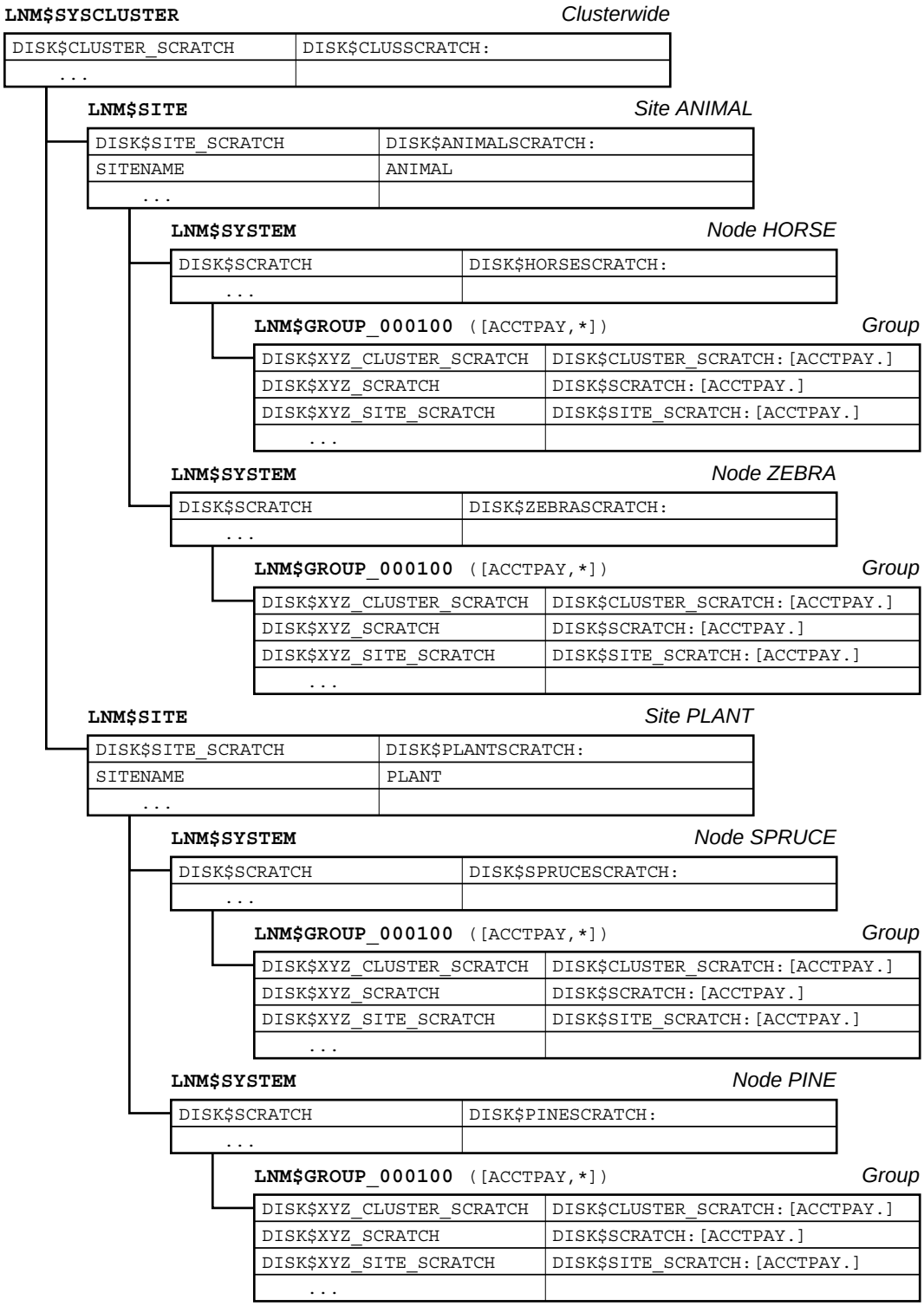

Figure 4 – Inheritance with rooted, concealed logical names.

Node	Translations according to Hierarchical Logical Name			
	Translation of SITENAME	Translation of DISK$SCRATCH	Translation of DISK$SITE_SCRATCH	Translation of DISK$CLUSTER_SCRATCH
HORSE	ANIMAL	DISK$HORSESCRATCH	DISK$ANIMALSCRATCH	DISK$CLUSSCRATCH
ZEBRA	ANIMAL	DISK$ZEBRASCRATCH	DISK$ANIMALSCRATCH	DISK$CLUSSCRATCH
SPRUCE	PLANT	DISK$SPRUCESCRATCH	DISK$PLANTSCRATCH	DISK$CLUSSCRATCH
PINE	PLANT	DISK$PINESCRATCH	DISK$PLANTSCRATCH	DISK$CLUSSCRATCH

Table 1 – Translations for the Accounts Payable Group ([ACCTPAY,*]) differ depending upon the contents of the hierarchical logical name table structure illustrated in Figure 4.

Developing a logical environment that is transparent across different systems requires care. In a clustered environment, the different member nodes must be fully interoperable, while still being appropriate for the different members and/or sites comprising the cluster. In both stand-alone and clustered systems, the logical environment must effectively embrace all operational issues without imposing unneeded presumptions of organizational structure or hardware configuration on any of the affected users (general users, developers, or system managers).

Issues

Different systems have their own issues, many of which are addressed through the judicious use of different elements in one of the login files, or the environment created by one of the elements of the STARTUP process.

Physical Machine Characteristics

Significant differences exist among system models. A small DS10 may be limited to an internal disk, and perhaps a small external storage shelf. A large GS1280 may have access to a complex SAN, as well as multiple local disks. Intermediate sized systems will be variations on the theme, with a hierarchy of storage, ranked by size, speed, latency, integrity, and cost.

Applications Impact

File placement depends upon use. Data files that are used throughout a cluster must be completely accessible throughout. On systems attached to a multi-site SAN, the straightforward choice for widely shared files is on SAN-mounted volumes.

Conversely, there are other files on the other extreme of the spectrum. Process-private temporary files are used only within a given image or process. A scratch file used by the SORT/MERGE utility is a common member of this class of files. Scratch files have no context or meaning outside of the moment, and need not be accessible to any other member of the cluster. Nor do they need backup, shadowing, or any other data protection scheme. If the process terminates for any reason, software or hardware failure, the files will, of necessity, be recreated when the process is restarted.

Often, temporary files are of substantial size. Placing them on remotely shadowed volumes, or volumes with full backup support, is a common source of overall system performance problems.

Physical Location

One of the strengths of OpenVMS is that it allows the programmer and system manager to generally ignore the actual location and configuration of mass storage. However, like any other virtualization scheme, this information cannot be ignored; performance and other issues merely move from one level to another.

It is tempting, but overly simplistic to consider the use of a SAN as a solution to all of the performance questions which bedevil large configurations. While SAN technology is quite effective, it cannot change the laws of physics. Some correlation between the function of mass storage and its location is extremely beneficial to system performance.

Implementation Aspects

These issues, concepts, and concerns may seem abstract or theoretical. This is far from the case. Examining the issues and concepts in the context of a particular example will help bring them into focus.

Consider a firm with a disaster-tolerant OpenVMS Cluster system located at two sites, with two systems at each site. To illustrate the full range of issues, let us assume that each of the four systems is different, ranging from an enterprise-class machine at the high end (e.g. a GS80 or GS1280), to departmental-class machines (e.g., ES4x) to small servers (e.g., DS10/DS20). A SAN is deployed at each site, and the systems take full advantage of the mirroring and shadowing facilities of OpenVMS and the storage controllers to provide mass storage.

User-Specific Configurations

The default value for **SYS$SCRATCH** created by **LOGINOUT.EXE** is the user's default directory, as contained in the UAF. This is far from an optimal decision for several reasons:

- the user's default directory will likely be on a volume that is mirrored (within a site) and/or shadowed (between sites). While conceptually simple, the reality is that there is an overhead associated with both local mirroring and remote shadowing. In the case of shadowing, the reality is that the inter-site interconnect has a finite bandwidth far less than that available either within the computer room itself or on the system's local interconnect.

- scratch files are frequently high activity files.

- scratch files may be very large.

- scratch files generally have little meaning outside of the particular process or job that created them.

A straightforward way to address this situation, while simultaneously using the configuration to best advantage, would be to relocate the scratch directory somewhere else. One obvious place is locally connected disks, namely disks connected directly to the individual system. Whether the scratch disks support all processes on the machine or a single user is irrelevant. The critical issue is access to an appropriate scratch area through the **SYS$SCRATCH** logical name as translated from within a user's individual process context.

So far, it seems quite simple. While this issue can sometimes be addressed at the level of an individual user, it is more appropriate to address it in a hierarchical fashion. Scratch areas

can generally be described on a group (department), set of departments, firm, or system basis, with only a small customization for the individual user.

Group-Specific Configurations

At the highest level of the scratch volume, create a series of directories, one for each department. These directories need to be accessible to all members of the group, either through rooted, concealed group logical name table entries for **DISK$SCRATCH**, or through rooted, concealed system logical name table entries for **DISK$GROUP_SCRATCH**, or other means.

In the **SYLOGIN.COM** file, we can redefine **SYS$SCRATCH** in the job logical name table (**LNM$JOB**) to point to the correct location.[14]

Thus, each group of users will seem to be pointing at their own scratch volume. For example, in the System or Firm Logical Name Tables we have definitions for **DISK$SCRATCH** as follows:

Group	Scratch Device	Value of **DISK$SCRATCH** in Group Logical Name Table
ITDevelopers	**DISK$NODESCRATCH**	**DISK$NODESCRATCH:[ITDEVELOPERS.]**
Accounting	**DISK$NODESCRATCH**	**DISK$NODESCRATCH:[ACCOUNTING.]**
Operations	**DISK$NODESCRATCH**	**DISK$NODESCRATCH:[OPERATIONS.]**

Table 2 – System/Firm definition variations in translation of DISK$SCRATCH

[14] The **ASSIGN/JOB DISK$GROUP_SCRATCH:[USERNAME] SYS$SCRATCH** command defines a supervisor mode name in **LNM$JOB**. The definition of **SYS$SCRATCH** generated by **LOGINOUT.EXE** is in executive mode. In most user situations, this difference is of no import.

If the dichotomy caused by having the supervisor and executive mode logical names pointing to different directories is an issue, it is possible to enable the **CMEXEC** privilege in the user's default privilege field in the UAF (but not in the authorized privilege field). **SYLOGIN.COM** will then execute with the **CMEXEC** privilege initially enabled. In this case, **SYLOGIN.COM** should immediately redefine the **SYS$SCRATCH** logical name in **LNM$JOB** as an executive mode logical name, and then downgrade the process by removing the **CMEXEC** privilege with the **SET PROCESS/PRIVILEGE=(NOCMEXEC)** command. Alternatively, a small privileged image could perform the same functionality with more restrictions.

If done properly, the preceding is not a security hazard. However, one should exercise prudence. In most cases, **SYS$SCRATCH** can be defined as a supervisor mode logical name with no ill effects.

The preceding may appear rather obvious, and indeed it is a rather simple example. However, suppose that as the system activity increases, we realize that the space and performance requirements of the scratch space for accounting have been underestimated. We decide to allocate a dedicated stripe set to servicing the large scratch space requirements of the Accounting group. With the above structure, only a single logical name needs to be modified, namely the definition of **DISK$NODESCRATCH** in the Accounting group's group logical name table as shown below.

Group	Scratch Device (value of **DISK$SCRATCH**)	Value of **DISK$SCRATCH** in Group Logical Name Table
ITDevelopers	**DISK$NODESCRATCH**	**DISK$SCRATCH:[ITDEVELOPERS.]**
Accounting	**DISK$NODESCRATCH1**	**DISK$SCRATCH:[ACCOUNTING.]**
Operations	**DISK$NODESCRATCH**	**DISK$SCRATCH:[OPERATIONS.]**

Table 3 – Translation of DISK$SCRATCH when System/Firm level definitions are overridden in the Accounting Department Group Logical Name Table.

The system does not need to be restarted, only the users in the Accounting group must be removed from the individual cluster member momentarily while the logical name is changed and the files migrated to the new location.[15] It is important to note that the impact of the change is very limited. Since the changes only affect the realization of the conceptual environment on the actual environment, only the system management group needs to be involved with this change.[16] Users will be unaffected, and in many cases unaware of the change, provided that they do not examine configuration-specific information not directly relevant to their applications.[17]

[15] Though not strictly required for scratch files, it is highly recommended. Copying the current contents of the old scratch device to the new scratch device during the changeover should not take long, and will prevent many problems for users who use **SYS$SCRATCH** as a storage place for transient files (e.g., test files generated during debugging sessions).

[16] An automatic procedure is useful for generating the group directory trees from a reliable roster of group members (e.g., an automatically generated and parsed listing of the system UAF).

[17] While users can use system services and DCL lexical functions to see the actual difference in their environment, there is no normal reason for them to do so. Done properly, the differences should be transparent.

Site-Specific Configurations

The preceding discussion addressed user- or group-specific scratch file locations. A similar technique can be used to identify a site-specific resource, such as a site-local scratch disk.

In its most primitive sense, a site-specific scratch location can be identified by the creation of a single name identifying the site that the system is associated with. **SYS$MANAGER:SYLOGIN.COM** (or the group logins, where appropriate) can then construct the logical name values appropriate for a particular site.

This approach allows the flexible provisioning of multiple levels of resources in a logically consistent manner. For example, three levels of scratch space can be made available to users and applications, each with different characteristics, as follows:

Name	Characteristics	Restrictions
DISK$SCRATCH	Machine local	Only available on a single system
DISK$SITE_SCRATCH	Site local	SAN connected, topologically local to each system
DISK$CLUSTER_SCRATCH	Clusterwide	SAN connected, shared with all cluster members

Table 4 – Scratch Resources by Location, Connection, and Accessibility

Company-Wide Defaulting

In a similar fashion, a hierarchical environment can be exploited to reduce the complexity and redundancy of applications. Suppose several sibling organizations share a hierarchically structured environment.[18] The same techniques traditionally used to support the differences between groups can be used to separate and support the parallel environments.

The parallel environments may represent subsidiaries or divisions of the same organization using common applications, different departments within a division, or customers of a service bureau. From an environmental perspective, the similarities far outweigh the differences.

[18] Conceptually, it does not matter if the actual systems involved are separate stand-alone systems maintained by common managers or applications developers, a single consolidated server, or an OpenVMS cluster comprised of many individual systems.

In creating a hierarchical structure to bridge department-level differences, we would use the group logical name table (**LNM$GROUP**). Users, who belong to different, yet parallel environments, require us to create a new category or level,[19] for example Firm (in this narrative, we will refer to it as **LNM_FIRM**).

Logical names reflecting the company-wide environment would be inserted into **LNM_FIRM**. In **SYS$MANAGER:SYLOGIN.COM** we would insert **LNM_FIRM**[20] in the logical name search path (**LNM$FILE_DEV** in **LNM$PROCESS_DIRECTORY**) between the group and system-wide logical name tables. Each group would be uniquely identified with a particular firm.[21]

It is admittedly simplistic, but this structure allows the creation of parallel application environments with minimal effort and minimal code differences between different branches of the tree.

The same process can be used to implement testing environments. It is admittedly expansive, but this approach can leverage the seemingly simple OpenVMS UAF, rights list, and logical name facilities to support large numbers of parallel development, test, and production environments for similar yet separate organizations on a single integrated OpenVMS Cluster system.

Company-Wide Constant Data

It is obvious that brief information specific to a particular sibling company can be stored in **LNM_FIRM**. Examples of such information are the name of the firm or the locations of company-wide resources or files.

Application-Wide Defaulting

The same principles that apply to individual subsystems apply to individual users or groups of users. The names of files and commonly used constants can be contained in an application-specific logical name table, and that name table can be inserted in one of the active search paths.[22] The benefits of this technique include:

---

[19] The logical name hierarchy in a baseline OpenVMS is cluster, system, group, job, and process.

[20] Paralleling the design of the OpenVMS logical name facility, **LNM_FIRM** would be a name located in **LNM$PROCESS_TABLE** containing a pointer to the name of the actual firm-wide logical name table for that particular process. The protection on **LNM_FIRM** must also be set appropriately.

[21] Each group belongs to an identifiable firm. Thus, it is possible to identify the correct firm-wide logical name table through a group-wide login script, the contents of the group logical name table, a rights list identifier, or a file in a common group-wide directory.

[22] If an entire community makes use of a particular application, it may make sense for that application's logical name table to be included at a higher level in the hierarchy than an individual user (e.g. the firm or group).

- easier maintenance – only one copy of the definition to use
- fewer logical names in the process or job logical name tables[23]
- faster logins as the command files defining the logical names need not be executed at each login
- faster **SPAWN** operations, as the voluminous process logical name table need not be copied to the sub-process each time a process is spawned.

Logical names used by different applications should not overlap. If the logical names used by pre-existing applications overlap, changing the sequence of name tables in **LNM$FILE_DEV** can be used to resolve the problem.

System Resources

The location of scratch space is but one example of an instance where the location or identity of system resources can be managed through the use of logical names.

Naming Conventions

Names that will be used globally should be named separately from names that are unique to a particular process or application. Care should be exercised to allow the same mechanisms to be used by different groups or ISVs. One possible way to avoid naming conflicts is to use the registered Internet domain names as the leading part of the logical name.

Summary

Presenting a conceptually consistent, although not necessarily identical user-perceived environment, is a powerful approach when implementing OpenVMS systems, whether stand-alone or as members of OpenVMS Cluster systems.

Hierarchical environments provide a powerful way to express the differences while retaining common elements. The source of the differences does not matter. The differences may be matters of mass storage configuration, as in the **SYS$SYSROOT** hierarchy used by OpenVMS itself, or the differences may reflect the management structure of the organization. Hierarchical environments allow the differences between disparate systems, and the differences in organizations, to be hidden from users and applications. The greater the disparity between the underlying systems or organizations, the greater the leverage of using

[23] Admittedly, memory consumption for logical name tables is not the concern for system performance that it once was. However, 200 logical names for each user on a large machine is still a potential performance issue when each of several hundred users defines a full complement of the names.

different, yet conceptually identical, environments to provide users and applications with a perceived identical computing context.

Using inheritance to dynamically instantiate logically identical user environments on dramatically different systems simplifies system management, reduces the cost of system management, and increases system availability. This method can be of even greater use to the end-user and system manager than to the base operating system and layered products.

Bibliography

HP OpenVMS DCL Dictionary: A–M, © 2003, Order #AA-PV5KJ-TK, September 2003

HP OpenVMS DCL Dictionary: N–Z, © 2003, Order #AA-PV5LJ-TK, September 2003

HP OpenVMS Systems Manager's Manual, Volume 1: Essentials, Order #AA-PV5MH-TK, September 2003

HP OpenVMS Systems Manager's Manual, Volume 2: Tuning, Maintaining, and Complex Systems, Order #AA-PV5NH-TK, September 2003

HP OpenVMS System Services: A–GETUAI, © 2003, Order #AA-QSBMF-TE, September 2003

HP OpenVMS System Services: GETUTC-Z, © 2003, Order #AA-QSBNF-TE, September 2003

OpenVMS Guide to System Security, Order # AA-Q2HLF-TE, June 2002

OpenVMS User Manual, Order #AA-PV5JD-TK, January 1999

OpenVMS Version 7.2 New Features Manual, © 1999, Order #AA-QSBFC-TE

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 1", http://www.openvms.org/columns/gezelter/logicalnames1.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 2", http://www.openvms.org/columns/gezelter/logicalnames2.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 3", http://www.openvms.org/columns/gezelter/logicalnames3.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 4", http://www.openvms.org/columns/gezelter/logicalnames4.html

Gezelter, Robert "The OpenVMS Consultant: Logical Names, Part 5", http://www.openvms.org/columns/gezelter/logicalnames5.html

Goldenberg, R, Kenah, L "VAX/VMS Internals and Data Structures – Version 5.2", © 1991, Digital Equipment Corporation

Goldenberg, R, Saravanan, S "VMS for Alpha Platforms: Internals and Data Structures, Preliminary Edition, Volume 3, © 1993, Digital Equipment Corporation, ISBN#1-55558-095-5

Goldenberg, R, Dumas, D, Saravanan, S "OpenVMS Alpha Internals: Scheduling and Process Control", © 1997, Digital Equipment Corporation, ISBN#1-55558-156-0

List of Figures

Biography

Robert Gezelter, CDP, CSA, CSE, Software Consultant, guest lecturer and technical facilitator has more than 25 years of international consulting experience in private and public sectors. He has worked with OpenVMS since the initial release of VMS in 1978, and with OpenVMS Cluster systems since their announcement in 1982.

Mr. Gezelter received his BA and MS degrees in Computer Science from New York University. He also holds the HP CSA and CSE accreditations relating to OpenVMS.

Mr. Gezelter is a regular guest speaker at technical conferences worldwide such as HPETS (formerly DECUS). His articles have appeared in Network World, Open Systems Today, Digital Systems Journal, Digital News, and Hardcopy. He is also a contributor to the Computer Security Handbook, 4th Edition, Wiley, 2002. Many of his publications and speeches are available through his firm's www site at http://www.rlgsc.com.

His firm's consulting practice emphasizes in-depth technical expertise in computer architectures, operating systems, networks, security, APIs, and related matters.

His clients range from the Fortune 10 to small businesses, locally, nationally, and internationally on matters spanning the range from individual questions to major projects.

He can be reached at <gezelter@rlgsc.com>.