OSF DCE SIG **J. Dilley (HP)**
Request For Comments: 44.0 **July 1993**

DCE CELL DIRECTORY SERVICE USAGE GUIDELINES

1. INTRODUCTION

This RFC presents a set of guidelines for the use of the Distributed Computing Environment (DCE) cell directory namespace by DCE clients and servers. By adopting common naming and usage guidelines, vendors and server developers will increase the likelihood that DCE-based software systems from multiple sources will not collide in the namespace. With consistent guidelines, individual users should be able to structure their profiles so that they can easily locate the servers they wish to use.

We began experimenting with DCE in May, 1991, in order to understand the benefits provided by this distributed systems technology. This document is based on the lessons we have learned in the past two years from building a set of prototype and product software using DCE and the CDS. Keep in mind that these guidelines are still evolving. The purpose of this RFC is to propose an initial set of guidelines so we may start moving towards consistency. Additional experiences, opinions, and guidelines are solicited.

2. BACKGROUND

The Cell Directory Service (CDS) enables DCE clients to locate compatible servers. In DCE, ‘‘compatible’’ is defined in terms of DCE *interfaces*: a server is said to be compatible if it supports (serves) an interface the client wishes to use (including interface version and object, if used).

Each server can support one or more interfaces described in an Interface Definition Language (IDL) file. The server *exports* its interfaces and its network location to the CDS directory namespace (hereafter referred to as the namespace). A client *imports* bindings for the server, allowing the client to make calls to that server. The client's act of *binding* to a server is a logical association that allows the client to make a *remote procedure call* (RPC) to that server.

For an introduction to DCE, see [Rose]; for an introduction to writing DCE applications, see [Shir]; for more information about RPC, refer to [OSF1] and [OSF2]. The paper that formed the basis for this paper was presented at the Karlsruhe DCE Workshop [Dill]. This paper assumes the reader is familiar with CDS. A brief introduction to the namespace hierarchy and layout are presented in the remainder of this Section. Section 3 contains guidelines and a discussion of some of the challenges of using CDS.

2.1. Directory Contents

The directory service supports a hierarchical namespace structure similar to that found in common file systems. It consists of a root directory and a number of subdirectories. Within each directory can reside zero or more leaf entries. These entries contain structured information about the servers in the cell. Each directory in the CDS namespace can hold four types of entries: directories, softlinks, clearinghouses and objects. Softlinks are pointers to other (leaf or directory) entries in the namespace. Clearinghouses are used by the CDS replication mechanism. Objects are used to hold the information for entries in the namespace; in specific, objects are used to store and retrieve server binding information. There are three distinct subtypes of objects used by RPC to store and retrieve server binding information:

(a) *RPC server entry.* This entry type stores the bindings of a server. If the server exports multiple bindings (for example if it has multiple interfaces) they can all go in the same entry. Each server entry has its own unique name in CDS.

(b) *RPC NS group.* A group holds an unordered collection of members which can be references to server entries or other groups. The entries in each subgroup will be recursively examined when a group is searched. For example, you might use an RPC NS group to hold information about all the print servers in the cell.

(c) *RPC NS profile.* A profile holds an ordered collection of elements which can be references to server entries, groups, or profiles. The elements are ordered into eight priority bands. When searching a profile, each server entry, group and sub-profile in a priority band are searched recursively for compatible server bindings. The bindings found in the highest-priority band are returned first to the client, followed by bindings from lower priority elements if the client continues to request more bindings.

In addition to the inclusion of groups and sub-profiles, each profile can also have one default profile that will be searched if the search of the current profile (and all of its contents) fails to locate a compatible server.

2.2. How CDS Works With DCE

When a DCE client wishes to look up a server, the client will need both the RPC interface identifier associated with a remote procedure it wishes to call and the name of a CDS entry from which to initiate the search for bindings. The CDS entry can be the name of a server entry, a group, or a profile. The client uses the CDS Name Service Interface (NSI) routines, given the interface identifier and CDS entry name, to search the namespace for bindings to a compatible server. The NSI will return binding information for a server, directing the client's call to the host where that server is operating. Naturally, a server must first export its bindings to the namespace so that clients can find it.

The client and server must agree on a CDS entry name in the namespace under which the server location information is stored. This name can be provided to the client and server in one of several ways, including:

(a) Hardcoded in client and server source code.

(b) Provided on the command line or in a configuration file.

(c) Computed at runtime using a standard format.

(d) Retrieved from the user's environment.

The server can have its binding information placed in the namespace either manually (statically) or automatically (dynamically). These alternatives are briefly introduced below:

(a) Manual registration means the system administrator is responsible for adding the binding information for a server to the namespace (with the aid of a script or administrative command).

(b) Static registration means the application is registered in the namespace when it is installed and appears in the namespace even when the server is not running.

(c) Automatic registration means the server itself is responsible for exporting its bindings into the namespace; no human intervention into the namespace is necessary.

(d) Dynamic registration means the application is registered and unregistered at runtime. This is usually coupled with automatic registration.

Upon receipt of the binding information from NSI, a client can make an RPC to the server associated with the binding. However, there is no guarantee that the information returned corresponds to a running server; the server may have terminated in such a way that it was unable to clean up its bindings in the namespace.

2.3. Namespace Layout

To receive the benefits of server location independence provided by CDS, each server should register its bindings somewhere in the namespace. But before you can register the server, you must decide where in the namespace to store the (unique) server entry and in which profiles or groups to include a reference to that server entry. These decisions require naming conventions that must be followed by the applications running within your cell.

2.4. Preexisting Entries

The CDS is configured by the `dce_config` script with a set of directories and entries already created. The directories are:

(a) `/.:/` — The local root of the cell namespace. Also known by the name `/.../`*`cell-name`*`/` where *`cell-name`* is the global name of your cell.

(b) `/.:/hosts/` — Contains subdirectories for each host configured into the cell.

(c) `/.:/hosts/`*`hostname`*`/` — The specific subdirectory for *`hostname`*, this directory initially contains bindings for DCE services that are running on host *`hostname`*.

(d) `/.:/subsys/` — A directory for holding DCE subsystems, including the directory `dce/`, as well as vendor-specific subsystems.

(e) `/.:/sec/` — The Security System junction. A junction provides the ability for other subsystems to join in the cell namespace. Access through this path, while apparently to the CDS, will result in a request to the DCE security subsystem.

(f) `/.:/fs/` — The Distributed File System (DFS) junction (also `/:/`). Through this path the DCE DFS appears to join in the namespace.

Some of the entries you will find are:

(a) `/.:/cell-profile` — The top-level profile for the cell. This can be used as the ultimate default profile for other profiles in the cell. It typically has no default profile, so any searches terminate after reaching `cell-profile`.

(b) `/.:/lan-profile` — The lan-level profile. This can be used to store LAN-local server information. It should be backed by a default profile like the `cell-profile`.

(c) `/.:/hosts/`*`hostname`*`/profile` — The local host's profile. Each host has its own private profile that can be used to hold information about servers running on the local host.

3. GUIDELINES FOR USING THE CDS NAMESPACE

A well-designed namespace must have guidelines defining how entries are named and where in the namespace they should be stored. What is most important is for the namespace to be internally consistent so users can easily navigate its structure and locate entries within it. In order for the namespace to remain consistent, vendors and software developers must agree to follow a set of guidelines that define how namespace entries are named, where they are located, and how servers access CDS. As the DCE namespace becomes more widely used, these guidelines will become more crucial to avoid collisions and other problems in the namespace.

The following Sections contain a set of recommended guidelines for the structuring and use of the namespace. These guidelines are labeled ‘‘G*g*’’ ($1 \leq g \leq 36$), for reference purposes.

3.1. Overall Namespace Guidelines

(G1) Store only server binding information in the directory namespace. While the namespace can be used to store arbitrary information, accessible via XDS, the CDS is not intended as a general-purpose database. Use of the directory namespace for purposes other than the storage of server bindings might impact the performance of the DCE cell as a whole. Therefore, to locate and access other information, use the namespace to store only the server bindings for a database server capable of returning that information.

(G2) Organize the namespace by creating a logical directory structure. Create directory namespace entries as you would files and directories in a file system. A well-organized namespace makes human searching and management easier.

(G3) Do not use the root of the namespace for storing arbitrary server entries, groups, or profiles. Keep the root of the namespace uncluttered for ease in searching. A few well-known profiles and directories in the root directory are acceptable.

(G4) Consider creating a `/.:/users/` directory under which each user has their own CDS ‘‘home’’ directory with write permission granted only to them. In this directory a user could store their personal profile, and register personal servers (see guideline G21). Providing users with their own area in which they can experiment prevents the need for them to have `cell-admin` privileges to use CDS. This enhances the overall security of the cell.

(G5) Have users register versions of experimental servers in their personal home directories for testing. If users are provided with their own environments in which to experiment, they should do so and not interfere with the regular operation of the DCE. Later, when applications are ready for wider beta testing, they can register their bindings in a more global test directory, such as `/.:/subsys/unsupp`. As they reach production quality, a server's bindings can be released into the appropriate location.

(G6) Choose sensible names for CDS entries using either the name of the server or a name that relates better to what the server does than its own name. Choose group and profile names that describe their purpose. For example, DeptX-Printers, or CAD-Profile. Avoid using random names in the CDS. In particular, do not use dynamically generated UUIDs[1] as CDS entry names. Entry names are meant to meaningful be humans; UUIDs are meaningful only to machines.

(G7) Avoid coding specific CDS entry, group and profile names in your applications. Doing this makes your application inflexible if a server's CDS entry name changes, or if its group or profile registration must change. Static registration also does not allow two instances of the same server to run within a DCE cell. Instead, allow the user to specify the CDS entry to use via the environment or command line (see guidelines G22 and G31). The namespace entries to use can be supplied to the server via a configuration file.

(G8) Register server entries in the most specific profile or group that makes sense. For example, register a departmental printer server in the department profile, not the cell profile. This avoids unnecessary searching in higher-level profiles and groups.

(G9) Create multiple CDS servers in your cell. Create replicas of important CDS directories on at least one other CDS server, preferably on two CDS servers so the environment is not halted by a single CDS server failure. See [Rose, Section 6.2.6]. Further guidelines for replication are required.

3.2. Guidelines For CDS Server Entries

(G10) Store only the bindings from a single server process in a CDS server entry, even if multiple servers support the same interface, or run on the same host. When multiple servers share a single entry, if any one of them unregisters its bindings, the NSI will remove all the bindings from the entry. This would orphan the other servers sharing the CDS server entry.

(G11) Register server bindings where they will not interfere with other servers, including other instances of the same server. One way to do this is to register their bindings under a name like `/.:/hosts/`*`hostname`*`/`*`server_name`*.

(G12) Have multiple instances of the same server on the same host use a suffix after *`server_name`*. For example, `server_01`, `server_02`.

(G13) Register servers also in a host-independent group or a profile. This group or profile would be used by clients to locate the server. Failure to do this will remove any location transparency since clients would otherwise have to specify the unique server entry.

(G14) Servers should register bindings in `/.:/hosts/`*`hostname`*`/`*`server_name`* by default if no CDS entry name was specified as described in guideline G7.

(G15) Clients should use `/.:/cell-profile` as a default profile if no host-independent group or profile name was specified.

3.3. Guidelines For RPC NS Groups

(G16) If server redundancy is important, use RPC NS Groups to store bindings for a set of identical servers. All group members should be server entries that have the same interface UUID. A client can perform a lookup in the group and continue searching until it finds a server that is up and able to satisfy its request.

The NSI will randomly select servers from the group and thus provide some rudimentary load balancing between servers. True load balancing, of course, requires knowledge of a server's current workload.

---

1. Universally Unique Identifiers. These are used by DCE to identify interfaces and other objects.

(G17) While profiles are currently more efficient than groups (see guideline 23), you should still use a group if the group abstraction more closely matches your application's needs — groups should be used to store a set of identical server entries, profiles should be used for other collections of entries.

3.4. Guidelines For RPC NS Profiles

(G18) Define a profile structure for your organization. Start with `/.:/cell-profile` being the profile of last resort (and with no default profile of its own). For example, consider creating department profiles that have the cell profile as their default. Consider creating project team profiles that have the department profile as their default.

(G19) Use profiles to hold collections of services that are logically related. For example, a department profile might hold references to all the database servers, phone number servers and print groups that are used primarily by that department.

(G20) Structure profiles to allow clients to locate nearby servers first, then to fall back on other servers found in increasingly higher profiles.

(G21) Provide users with their own profiles in which they have write permission, so they can define their own specific profile and group search paths. Supply them with a default profile pointing to the next level up in the profile hierarchy (for example the project team profile). By providing users their own profile and home directory, it becomes easier for them to experiment with CDS, and with the DCE, without interfering with other users.

(G22) Users should specify their personal profile as the default for namespace searches using the environment variable `RPC_DEFAULT_ENTRY`. This variable is used by clients that employ the automatic binding method.

(G23) When registering a collection of server entries with different interfaces, use profiles instead of groups. Profiles contain the interface UUID of the server registered in the profile element; groups do not contain this information. This makes searching profiles more efficient.

3.5. Server Guidelines

Servers have the responsibility for exporting their bindings to the namespace so clients can find them, but at the same time keeping the namespace clean.

(G24) Servers should register their bindings dynamically as the final part of their initialization, when they are ready to receive and process RPC requests.

(G25) Each server should register all its interfaces and associated bindings in a single (unique) server entry to conserve the number of CDS entries. If not stored together, the server would have to register its bindings in multiple entries on initialization and unregister them from all those entries on termination.

If a server supports multiple interfaces it may be appropriate to register a reference to its server entries in different profile and group structures. For example, a server may export its entry to a profile for its operational interface, as well as to a group that holds references to all servers that support a particular administrative interface.

(G26) Servers should unregister their bindings before graceful termination to avoid leaving stale bindings in the namespace. Servers that terminate abnormally can leave stale bindings, however. If a client attempts to use a stale binding its call will eventually time out and raise an exception in the client (the default time-out period is 30 seconds; see the `rpc_mgmt_set_com_timeout()` manpage).

(G27) Servers should also catch and handle asynchronous signals, generated by keyboard interrupts and the `kill` command issued against the server process, to allow graceful termination.

(G28) Servers should unregister their server entry from any groups or profiles before unexporting bindings from their server entry. If they fail to do this they will leave a dangling reference to that server entry in the group or profile. The NSI will effectively ignore dangling references, but they do have a slight performance impact, as well as an impact on namespace management.

(G29) Servers should refresh their credentials with the DCE Security Service before they expire so they can later unregister their bindings from the namespace. This will avoid the situation in which stale bindings remain in the namespace because a server no longer had the security credentials required to remove its bindings when it terminated.

(G30) Servers should store only relatively stable but non-static information in the CDS to avoid overfilling it or thrashing it. If a server is only going to be up for a short time, or only to service a single client, then do not store its bindings in the namespace.

3.6. Client Guidelines

Clients should always anticipate the possibility of failure and deal with failures appropriately. Poorly designed DCE clients will fail if they attempt to call a server using stale bindings retrieved from CDS. Clients should be written to be robust by checking error status and watching for other types of failures.

(G31) Consider providing a client command line option to allow the user to specify the server protocol sequence and host addressing information. Specifying a server by host address is less flexible than using CDS, but providing the option may allow the client to function even if the CDS is down or performing poorly. See Section 3.8.

(G32) DCE clients should watch for exceptions being raised by the RPC runtime due to stale or defunct binding information. (If the `comm_status` attribute was used, the client should check the status return code for a failure.)

(i) If a failure occurs following an RPC, the client should first reset the endpoint in the binding and retry the call. If the server went down and came back up again on the same host, this will locate the new instance.

(ii) If resetting the binding does not work, the client should retrieve another binding handle from the namespace and try to contact another server.

(iii) If all bindings have been tried and no RPC was successful the client should print an informative error and exit.

(G33) To avoid writing code that implements the above cycling through bindings, use the automatic binding facility provided by DCE RPC: it will do this for you. Automatic binding also maintains the local procedure call paradigm as much as possible. Note, though, that there are some restrictions automatic binding places on your application, such as the inability to specify security parameters. For more information see [OSF1, Section 13.5].

(G34) If a server (or the server's host) has gone down and the client looks for more bindings in the namespace, it might get another binding that refers to the same server. This is likely to happen if the server has registered itself using multiple protocol sequences. In this case, a call using the new binding handle will fail also. An intelligent client application could check the binding retrieved from the namespace against the one that just failed; or it could simply continue to look for more bindings.

3.7. Performance Guidelines

Performance of the CDS is an important consideration when writing distributed applications. However, namespace performance is particularly dependent upon the underlying implementation. For now the recommendation is to be aware of the current characterizations of namespace performance [Mart]. More formal guidelines will be investigated for a future draft of this RFC. Please submit your performance guidelines to the author of this RFC for inclusion.

3.8. Namespace Management Guidelines

This Section still needs development — namespace management is still evolving and depends upon the increasing deployment and use of the CDS. What follows are some initial suggestions.

(G35) In addition to writing a robust server, you can construct a management application that checks the CDS for accuracy. An example of such an application is one that searches CDS for all servers registered under a given interface identifier, using the same method used by clients to locate that server. If the management application detects that any server is not running, it can remove that binding information from the CDS. This decreases the likelihood that future clients will have to perform failure recovery.

(G36) Be very careful when cleaning up CDS. When security credentials are required there are times when you should not remove stale bindings or dangling references in groups or profiles: the administrator can create a server entry in a secured directory that allows a specific principal write permission. If that server entry is removed, the principal could no longer add the server bindings to the entry.

4. ACKNOWLEDGMENTS

This document is the culmination and summarization of work by the Distributed Systems Architecture Team. This work owes thanks in particular to Deborah Caswell, Christopher Mayo and Jeff Morgan for their development assistance, reviews of this paper, and helpful suggestions. This work also benefitted significantly from review by Paul Smythe of HP's DCE core team, and the NSA Distributed Computing Performance Team, Peter Friedenbach, Rich Friedrich, and Joe Martinka.

REFERENCES

[Rose] W. Rosenberry, et al, *Understanding DCE*, O'Reilly and Associates, Inc., 1992.

[Shir] J. Shirley, *Guide to Writing DCE Applications*, O'Reilly and Associates, Inc., 1992.

[OSF1] *OSF DCE Application Development Guide*, Open Software Foundation, 1991.

[OSF2] *OSF DCE Application Development Reference*, Open Software Foundation, 1991.

[Dill] J. Dilley, ''Practical Experiences with the OSF Cell Directory Service'', Networked Systems Architecture, Hewlett-Packard, International Workshop OSF DCE, Karlsruhe, Germany, October 1993.

[Frie] P. Friedenbach, et al, ''Performance Characterization of the DCE 1.0.1 RPC Name Service'', Networked Systems Architecture, Hewlett-Packard, Hewlett-Packard Company Document Number NSA-92-018, December, 1992.

[Mart] J. Martinka, et al, ''A Performance Study of the DCE 1.0.1 Cell Directory Service and Implications for Application and Tool Programmers'', Networked Systems Architecture, Hewlett-Packard. International Workshop OSF DCE, Karlsruhe, Germany, October 1993.

AUTHOR'S ADDRESS

John Dilley
Hewlett-Packard Company
19410 Homestead Road, Mailstop 43UA
Cupertino, CA 95124
USA

Internet email: jad@nsa.hp.com
Telephone: +1-408-447-2419