Distrib. Syst. Engng 1 (1993) 19–28. Printed in the UK

The distributed computing environment naming architecture*

Norbert Leser†

Open Software Foundation®‡, 11 Cambridge Center, Cambridge, MA 02142, USA

Abstract. The OSF® Distributed Computing Environment (DCE) constitutes a set of technologies which have been selected through an open acquisition process and integrated into a coherent DCE architecture. This paper discusses the architecture of DCE while focusing on the architecture of the name service as one of the key elements.

The notion of the *cell* is introduced and described. The cell is an essential property of the DCE architecture. While providing a means for modelling a distributed system, cells go beyond being a formalism for describing the system. This paper elaborates on the role and purpose of the cell, the semantics of the cell namespace, the placement in a large worldwide spanning system, the implications of security in a distributed system, and other related architectural properties of DCE. All this will be discussed in conjunction with other architectural approaches of modelling distributed systems.

The research and engineering work on this paper is based on a leading involvement with this project during the last four years. It includes studies and evaluation of worldwide available technologies and research projects, architectural, design and developmental work on these technologies and experience with further usage of this system, in particular the integration of the object oriented distributed management technology, using DCE Naming and Security Services.

1. The key elements of a distributed system

The DCE architecture was determined and developed by analysing currently available technologies and by defining the inherent properties of distributed systems [1, 2]. Essential requirements are the needs for integrating networked systems into a coherent distributed systems environment and for solving system growth from homogeneous parallel processing systems into distributed systems [3].

The architecture of a distributed system cannot make any assumptions on homogeneity due to the diversity of the distributed environment. One has to deal with *heterogeneity* on all levels, from different machine types, various networks and communication protocols, up to service levels. Multipurpose services may be sufficient for most applications but for various reasons, be it competition or be it the need for especially tailored services, the distributed systems architecture must be capable of adopting diverse technologies.

Another significant area which needs attention in architecting the distributed system is the need for *unlimited growth* of the system. The entire connected distributed system must be sufficiently scalable worldwide and may have to deal with millions of nodes.

1.1. The core distributed services and the role of naming

In creating an architecture for the distributed computing environment we were faced with the requirement that the traditional modelling of single systems on one hand and purely network and communications technologies on the other hand are not sufficient to provide a coherent view of the distributed system. Thus, we developed an architecture which provides for an integration of a well defined set of *core services*, creating an infrastructure for programming and using the facilities of the distributed system. These services include technologies for remote procedure call (RPC) based communication, security, a provider for a truly distributed time, and a naming system [4, 5]. Figure 1 shows the relationship of the DCE technology components.

Naming was always considered to play a central role in meeting the goal of providing transparency and homogeneity to the consumers of the system by hiding the inherent complexity, heterogeneity, and diversity of the distributed environment. The naming architecture

† Internet e-mail: nl@osf.org

0967-1846/93/010019+10$07.50

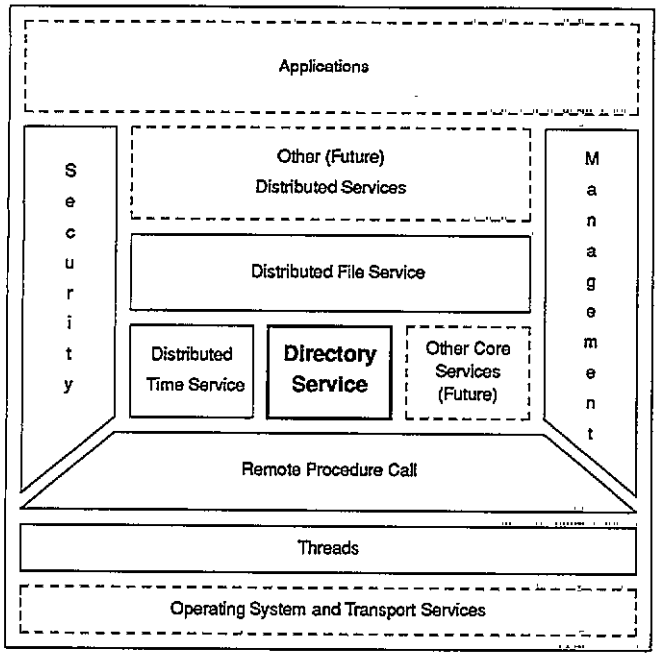

Figure 1. DCE component architecture.

must be able to cope with the broad variety of entities in the system as well as with the unlimited size and growth.

One may imagine how different this can be. Already after only 12 years of Internet the Domain Name Service (DNS) just hit the 1 millionth registered node. Consider that these nodes are only network addressable hosts, and other to be named and addressed entities such as distributed services are not managed with DNS. Furthermore, the increasing settlement of object oriented modelling and programming in the distributed environment will have to deal with the creation, and deletion, of objects at a much higher rate. Even if only a small part of these objects are permanent and need to be registered, it becomes obvious that the growth rate will be even higher than currently experienced.

While the entities mentioned above, network nodes, services and application objects, are essential for architecting naming in the distributed environment, there are a number of other types of objects such as persistent traditional directory entries (countries, organizations, persons, mail addresses) or data entities like files which should fit coherently into the naming model.

1.2. The architectural requirements for naming

It appears obvious that this wide variety and number of objects cannot be sufficiently dealt with by just one technology. But it is also obvious that the usage of the distributed system requires as much uniformity as possible in accessing the entities.

The first distinct requirement one can observe for name services is the *access model.* Most commonly, name services are used to look up the desired information based on a given name for an entity, where names are either fully distinguished names or names relative to a certain starting point (context) in the namespace. The access model also implies that updates and changes are usually much less frequent than lookup operations. For other special purposes, more complex queries which involve extensive filtering and searching or requirements for a very frequent update rate may become dominant.

The other area where name services behave distinctly is in the *organizational model* of the namespace. While usually a hierarchical tree organization is sufficient, in other cases a flat organization with access through hash tables, or topographical views and directed graphs which, for example, account for multiple inheritance, may be preferable. Also, the desired complexity and flexibility of the structure of the organization may vary. Systems which provide for sophisticated schema management need to be considered.

In this context, the complexity and *semantics* of entries varies widely. For some applications it may be preferable to have just basic attribute values associated with entries while for other applications structured attribute value assertions (type–value attribute pairs) are required.

Other requirements for the name service are reliability, manageability, and security. Since these usually conflict with performance and scalability, one has to be able to balance them according to the particular purposes. Replication, for example, the technology to provide for reliability and availability of the name service information, can be provided with varying degrees of weak to strong consistency. While in some cases lookup failures and retries are cheaper and more acceptable than in others, no single solution can be optimal.

1.2.1. Can federated naming solve the problems? An uncountable number of technologies which deal in one or the other way with naming have been developed and have matured. These range from complex directory services to location brokers, network information systems, object inheritance services, and file systems. Various naming facilities coexist in the network, and will continue to do so. This is due to the outlined technical reasons and because of political considerations such as corporate and national policies, suppliers interests in particular market segments, and commercial and financial security. Therefore, there is an urgent need in a distributed environment to provide for a coherent integration of these technologies.

Several approaches for solving these problems have been discussed to a large extent in the community and have been accompanied by several experimental implementations. One of the inspiring architectural discussions was driven by the Advanced Networked Systems Architecture (ANSA), developed at Architecture Projects Management Limited in 1989 [6] This work introduced the notion of *federated naming* as a fundamental concept of modelling naming. Federated naming allows for coexistence of distinct and diverse name services and provides translators

and gateways to communicate beyond the system's boundaries. Rather than approaching a global naming model, the unambiguity of this naming model is based on context relative naming in federated, heterogeneous domains. Interworking is achieved through bilateral cooperation.

Cooperation of federated naming domains, which is more precisely described as interconnection rather than interoperability, is achieved through gateways which wrap and unwrap the appropriate reference information. The wrapper provides an envelope which details the context and mapping of higher level domains. Directed graphs of name contexts rather than the ordering in a hierarchical tree structure is the basis for this structural model.

Federated naming, though, does not completely address the one crucial issue in a distributed environment. It is necessary to mask the complexity and heterogeneity of the system by providing as much transparency and homogeneity to the users as possible. Transparency is certainly not a fixed measure, but final consumers of the system usually do not want to have to deal with differences in the underlying technologies, while programmers or administrators need to have more flexibility to fine tune and tailor the system. However, since pure federated naming does not prescribe basic semantics and certainly not syntaxes, interoperability is limited and requires the knowledge of all involved instances.

Strictly centralized name system such as the X.500 directory system [7] is the orthogonal approach. It does not provide a means for incorporating other naming systems, but it seems to be much more appropriate and intuitive for a user to think in and deal with hierarchically structured organizations.

2. The notion of name spaces

In refining the naming architecture for DCE we intended to combine the compelling features of both approaches, the federated naming and the centralized global naming model, into one coherent architecture. Instead of addressing the bilateral issues between heterogeneous naming domains, which are actually instances of particular services, the scope of the naming system was defined by discrete namespaces with its distinct properties in the distributed environment. These *composite namespaces* are an abstraction of particular name services or resource managers† describing their syntactical and semantical rules.

The primary namespaces are the *global namespace* and the *cell namespace* which are connected via a gateway, the Global Directory Agent (GDA).

† In many instances the term resource manager is being used since the services supporting these composite namespaces are not necessarily name services in the generic sense.

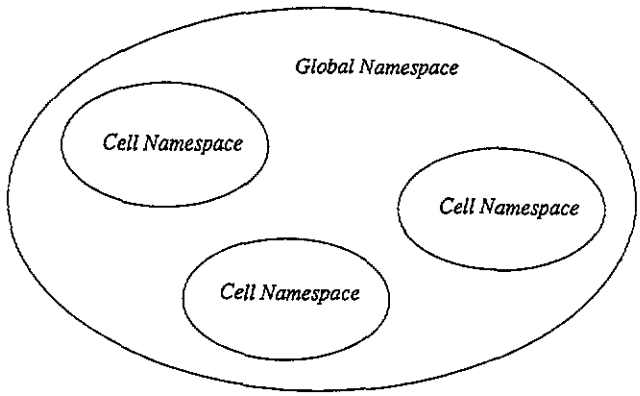

Figure 2. DCE namespaces.

This decomposes the namespaces into a hierarchical organization (see figure 2).

Although this paper primarily addresses the DCE naming architecture, specifying the global and the cell namespaces, the service instances, the DCE Global Directory Service (GDS) and the DCE Cell Directory Service (CDS) will also be discussed. It should be noted that the current implementation of the interfaces and protocols already provides the core features, but some of the functionality outlined in this paper are under development and will be noted accordingly. For instance, the refinement of the junction protocol and the specification of a uniform API are currently in the design phase‡.

2.1. Assumptions for the cell name space

The cell name space is the basic entity which describes an organizational unit of DCE, an administrative domain. Policies and various degrees of protection levels, the security realm is determined by these cell boundaries. Administrative regions and domains may share the same boundaries or they could be organized as substructures.

Typical considerations for the requirements are:

(i) The intra-cell connectivity is usually higher than the degree of connectivity with systems located outside the cell. LANs rather than WANs are the primary communication channels of systems inside the organizational boundaries of a cell.

(ii) There is implicitly more trust between a principal and an object within the cell than those across cell boundaries.

(iii) The cell naming environment is unified regarding the semantics and syntaxes applied to this namespace.

‡ In the meantime, after the first publication of this paper at the OpenForum Conference in Utrecht (November 1992), an X/Open driven initiative led to a multi-national project for defining the *X/Open Federated Naming Service* (XFNS). XFNS will include the specification of an API and protocols for federated naming. Several leading computer companies are participating in this project whose goal is to produce specification and prototype of both interface and protocols. XFNS will complement the currently available DCE naming technology. It is equivalent to the described design.

2.2. The global name space

To provide uniformity and worldwide interconnectivity, cell namespaces are always part of and integrated into a global namespace which provides at minimum the location service for foreign cells. The global namespace is represented by a defined small set of globally available centralized name or directory services. Supported references of the global directory services are the Internet Domain Name System (DNS) and the X.500 directory system.

Although a single directory service technology would not be a sufficient and acceptable solution for the entire namespace, it appeared to be sensible to use the few dominant and matured central directory services as facilities for the global namespace. The Internet Domain Name System (DNS) and the IEEE X.500 directory system are clearly the preferred name services which provide worldwide access and interoperability. Both are supported as reference in the DCE implementation. It is likely that X.500 will be the dominating system in the future. Nevertheless, the DCE naming architecture does not restrict the instantiation of the global namespace to just one directory system. Multiple global directory systems can coexist and provide interconnectivity between cells† which are represented by different global directory systems.

Additionally, the architecture allows a single cell to be registered with multiple global directory services, however, it is still debatable whether this is desirable. The current implementation of DCE restricts the cell registration to one instance of the global directory service.

The registration of a cell in a global directory service differs for each particular service, depending on registration policies and semantics. In X.500, two attributes in the directory entry composing the cell name (CDS-Cell and CDS-Replica) contain the cell identifiers and protocol sequences necessary to access the instance of the cell directory service. Analogous in DNS, cell names and other relevant information are associated with resource records.

2.3. The Global Directory Agent (GDA)

The connection between the global namespace and the cell namespaces is handled through a Global Directory Agent which acts as a gateway. The primary function of this gateway is to resolve the global part of a given pathname by issuing a referral protocol exchange to the appropriate global directory service. A successful path resolution returns a set of binding handles for the directory service in the targeted cell. The second task of GDA is its function as protocol translator.

While the cell namespace is RPC protocol based, the global namespace is considered to use the respective native protocols such as Internet or the OSI stack.

The initial purpose and design of the GDA is its function as trader. It delivers the binding information of the remote service provider to the service requestor. It is conceivable that a GDA could also act as a gateway for performing other directory service operations in the global namespace such as lookup on arbitrary attributes or modification of entries. This is an area which requires some more investigation; in particular the expected benefits need to be defined.

† This can be achieved since cells, and more generally each DCE entity, is uniquely identified by a Universal Unique Identifier (UUID) which is guaranteed not to be duplicated or reused, even if generated in independent and disconnected systems. The UUID is an automatically generated 128-bit identifier using IEEE node identifiers and time stamps for its generation and therefore is unique across both space and time. It does not contain the location information of an object. The UUID, with its fixed length, is not infinite, but it is large enough to allow the generation of more than 10^6 objects per second on each node over the next 10^4 years.

3. The properties of the cell namespace

The DCE Cell Directory Service (CDS) represents the cell namespace. It plays a key role in the DCE naming architecture. While the cell is the administrative domain in the distributed computing environment, CDS is particularly suited to provide the following services.

- Providing the information for distributed applications to locate and bind the service requestor (client) with the service provider (server). These operations which generate and obtain the binding information are also called *trading*.
- Connecting multiple heterogeneous naming domains into one coherent environment. This includes the resolution of references to the global namespace and *junctions* to descendent namespaces which can be subcells of a hierarchy of cells or namespaces represented by distinct services such as the file system. CDS can essentially be considered as the central instance for connecting composite namespaces.

Figure 3 shows the organization of a cell namespace. The composite namespace within the cell namespace are accessed through the junction points which are actually directory entries in CDS. Directories and objects registered in CDS (connected by solid lines) are primarily used for the trading purposes. For instance, *subsys* decomposes a subtree containing protocol towers (see below).

3.1. DCE cell directory service

Although the design of the cell directory service is explicitly tailored to meet the naming needs for the distributed system itself, to act as trader and as a service which integrates the various namespaces, there is no reason why it could not be used as a general purpose directory service as well. Its semantic is rich enough to support arbitrary attributes and application-specific naming needs. It is the application design which will

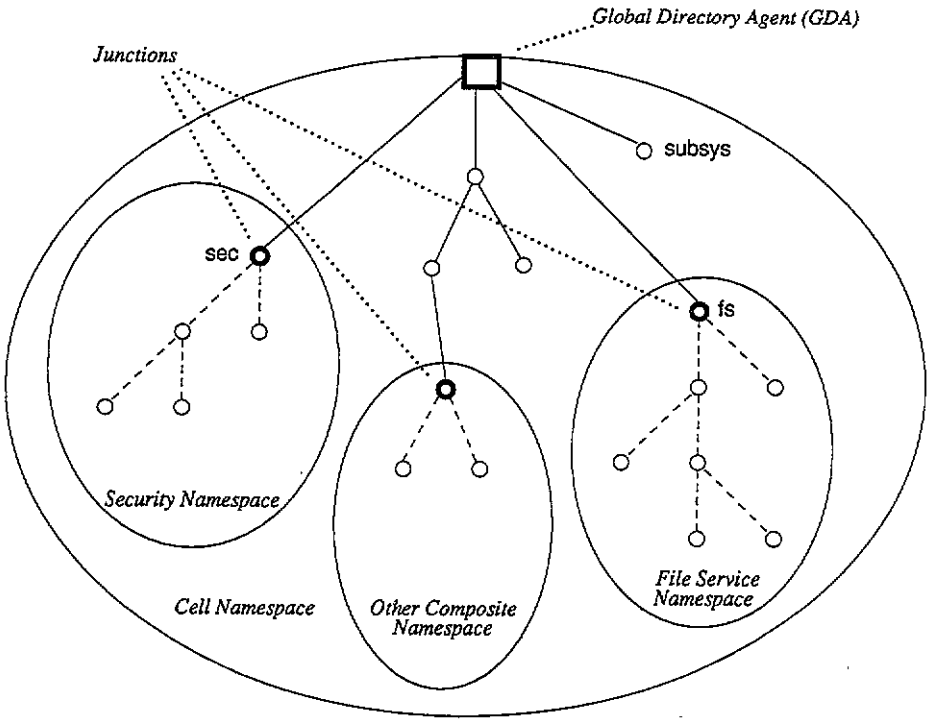

Figure 3. Decomposition of the cell namespace.

determine whether this is sufficient or whether special purpose naming services may be more applicable.

In contrast to a universal directory service, the name service supporting the cell namespace is tailored to meet the special needs in the cell environment. Key requirements are protection, reliability and availability, lookup speed, ease of management, and sufficient communications protocol.

While the binding information retrieved from the cell name service is essentially a reference pointer, the *authentication* and *authorization* verification of a client/server application is a function of the respective applications. In DCE, the name services are not part of the trusted computing base (TCB), and therefore, cannot guarantee integrity of the contents of the information which has been generated by the service providers. This raises interesting security considerations for name services which have similarities to databases. Not the (name service) database but the services which are permitted to access and modify its contents are the trusted entities. Services should always rely on the DCE Security Service, which is part of the trusted computing base in DCE. Assuming the usage of the mutual authentication and authorization capabilities of the DCE Security Service, denial of service attacks would be the worst case scenario with compromised Cell Directory Services. This is usually acceptable, but if security sensitive environments require protection against denial of service attacks as well, their name services must explicitly be integrated into the TCB (for example, installing them on secure hosts).

Availability and *reliability* of the cell directory service information is one of the other requirements with high priority. One can make the observation that the distributed system can only be as reliable as the cell directory system. While availability is usually achieved through replication of complete databases or subsets of them. The degree of consistency depends on the underlying replication technology. For the purposes of the cell directory service it is usually not necessary to provide a strong transaction based replication but the probability of acquiring reliable data should be very high. The system must be capable of detecting mismatches and must provide the flexibility of invalidating and refreshing cached information.

The ratio between *update* and *lookup frequency* is an important factor in the name service layout. A well designed distributed environment must be designed in such a way that bottlenecks in large scaled systems and additional overhead introduced by distributing applications is kept to an absolute minimum. Crucial areas here are additional computation and traffic for security operations, computation for converting data representations, the speed of the physical links, and in particular the latency during association setup and binding. For the purposes of the cell name service, an optimized high speed retrieval of registered binding information has a clear priority over the registration operations, which is an administrative activity and performed much less frequently.

The communications protocol for CDS is *RPC based.* DCE RPC provides the desired degree of transparency independent of the underlying communications structure. Both Local Area Networks and Wide Area Networks (primarily used for inter-cell communication) are efficiently supported. The

semantics of the DCE RPC protocol provide the required reliability and guarantees of delivery. Both a security interface for providing mutual authentication and an interface for trading are inherent in the RPC support.

3.2. Trading

Trading in the client/server model is the notion of establishing the association between matching service requestors and the service providers. This process of associating a client to its server peer is of a dynamic nature in the distributed system and it is essential that this is kept as transparent to applications and users of the system as desired. Trading involves a number of considerations.

- Clients and servers may have been developed independently, at different times and sites, by different developers.
- Multiple instances of servers may offer the same services, and they may operate on the same or on distinct objects.
- The number and sites of clients is not predetermined.
- The selection of servers can depend on factors such as load, network latency and bandwidth, availability.
- Servers may crash or hang, and a rebinding may be appropriate, assuming the context has been preserved.

Trading is a generic service which can provide arbitrarily complex information and sophisticated operations. Algorithms can be applied to trading for very dynamic binding behavior, depending on a number of parameters to allow for load balancing or process migration in the distributed sense; or the binding to methods and operations on objects can be supported by following inheritance rules. Such elaborated systems can be built using the DCE naming architecture and atop of the DCE technology implementation. This paper primarily outlines the more primitive operations and facilities necessary to provide for automatic binding, transparent to applications and users.

For resolving the trading issues, the client/server model uses object oriented concepts and treats server instances as objects. Servers *export* an *interface* or a set thereof which specify the remote operations which can be performed on these servers. These interfaces are uniquely and unambiguously identified by Universal Unique Identifiers (UUIDs), and the trading operations essentially resolve the match of interfaces supported by clients and servers. Servers therefore export their interfaces to the cell directory service.

Since instances of servers may operate on one or multiple objects, a further determination of the target in the client/server connection allows for additional *object UUIDs*. For instance, a printer server may support various different printers such as postscript or line printers. While the interface UUID defines the printer server, object UUIDs optionally can further detail on which printer a certain job shall be performed.

The client, which is the service requestor, on the other hand, *imports* the *binding information* for establishing the association with any available and matching server from the cell directory service. The algorithm for selecting one of the available servers can be predetermined or specific to a particular instance of an application.

The binding information obtained from the name service contains the location, the network address of the server, and the set of matching protocol stacks, since it is possible that servers may have exported multiple sets of supported protocols. These protocol sets are actually stored as *protocol towers* in the name service. The number of levels of these towers certainly varies, but a tower must sufficiently define the matching set of protocols. A protocol tower could, for example, contain protocols for the data representation, RPC, transport, and the network, in addition to the interface UUID.

For navigating through the namespace and finding the right set of servers, the trading facility provides for three classes of entries: server, group, profile. *Server entries* contain the protocol and addressing information of one server instance, while *groups* are an unordered set of equivalent server entries (servers with common interface and object UUIDs). Groups can be nested.

The *profile* entry in the name service contains a collection of profile elements. These elements are determined by a single interface and contain an ordered list of members which can be servers, groups, or profiles. The ordering of these providers for the particular interface is achieved through priorities. A profile may also contain a default profile element, which is a pointer to another (default) profile which will be searched if no compatible binding could be found in the profile. This allows for customizing the ordering of the search for a compatible server. This could follow topological views or other means of structuring the organization.

The outlined model demonstrates how the cell directory service provides a means of hiding the complexity of the distribution for the consumer of applications. Administration and mangement of these entries, such as profiles, can be centralized within cells but it is also permitted and possible to overwrite certain default setups.

As clearly as the final consumers of the system must be supported, the programmers of distributed applications must also be provided with instruments able to handle the complexity but also flexible enough to develop applications which can dynamically be installed and configured. A special purpose *application programming interface* (API) must be sufficient for exactly performing the set of name service operations necessary for the binding or trading operations. Only designated system administrators should be required to further manage the name service setup and only those application writers who want to use the cell name service for purposes beyond the trading operations should be required to use a more sophisticated general purpose application programming interface.

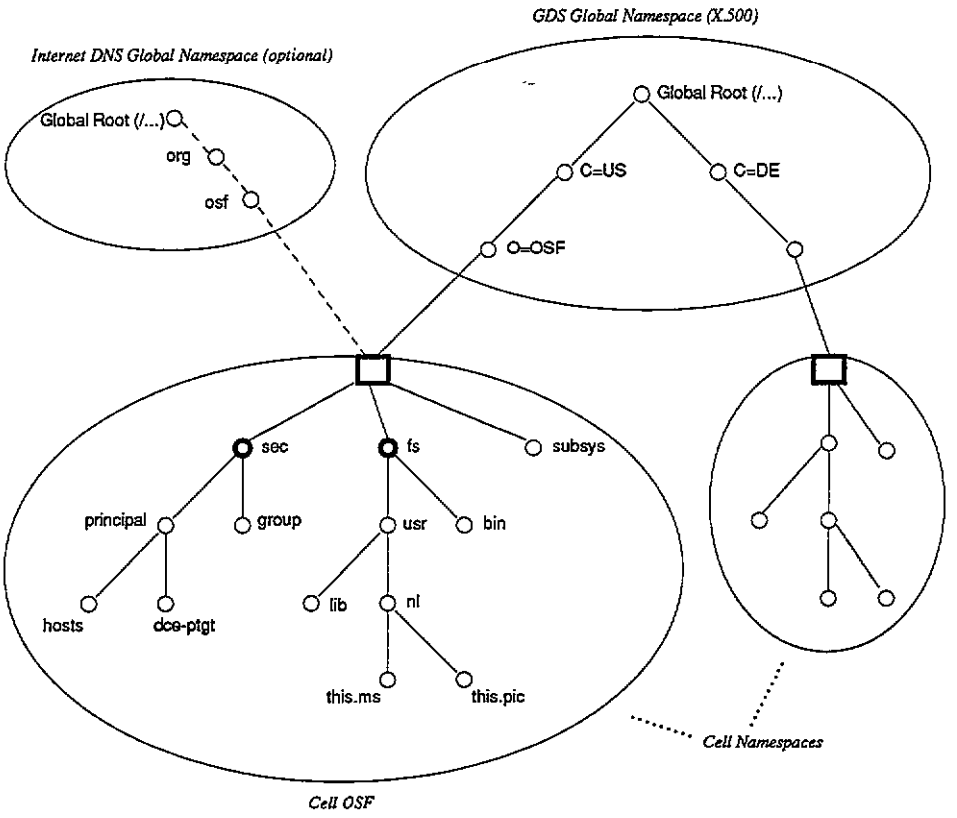

Figure 4. Example of a complete DCE directory system.

3.3. Junctions

We can make the assumption that a generic naming service has properties such as slowly changing namespace and weakly consistent replication. While this may be suitable for objects which reside in global or cell directory services, there are a number of objects in the distributed computing environment which have different naming characteristics. This requires a mechanism to incorporate composite namespaces into the global namespace, or more precisely, the cell namespace.

All object names which are exported to the cell namespace must be resolved via a single name resolution protocol. This includes the capability of deriving from the name the object location and potentially a minimal set of other information about the object. DCE introduces a DCE RPC based junction protocol which currently provides for the incorporation of the security and the distributed file service namespaces into the cell namespace†. Other examples for special purpose resource managers are inheritance trees for object oriented modelling or services for transitive, very short lived entities such as processes. The junctions allow clients to resolve the CDS portion of the name and pass the residual part of the name to the resource managers of the respective composite namespaces. While these junctions are currently specific to each resource manager, DCE will provide a generic junction protocol and the appropriate interfaces to support uniform cross resource manager operations (see figures 3 and 4).

In order to integrate other composite namespaces into the cell namespace, the junction protocol requires the resource managers to support at minimum a set of functions to resolve the residual components of the pathname by using a defined referral mechanism. The binding information of the target object and possible attributes, defined in an attribute schema, will be the result of the junction operations.

It is conceivable that future protocol extensions and agreements on a low level naming interface will allow for rudimentary operations across name services and resource managers. These may be operations such as add, update, delete, and list objects and attributes.

The syntax of the pathname passed to the junction protocol must comply with the DCE name syntax described below. As outlined, components of this pathname (slash separated) may contain a name structure according to the semantics of particular resource managers.

3.4. Hierarchy of cells

Cells are administrative domains which are essentially determined by the security policies of organizations; thus cells are usually equivalent to realms in the security space. Communication across cell namespaces requires the establishment of additional trust relationships involving the cell-based certification

† The Global Directory Agent (GDA) is a special form of junction which provides the capability of resolving the binding for a foreign cell to the local Cell Directory Service. The referral of a successful GDA operation points to CDS in the targeted cell.

authorities. Independent of the employed encryption technologies, whether they are public or secret key based, it is not likely that independent organizations will agree on an external agent controlling the global certification authority.

Matters are completely different within enterprises and organizational boundaries. For managerial or other organizational reasons it appears that enterprises want to be able to manage subdivisions separately, but in contrast to inter-cell relations apply central authorities for policy determination, administration, and security. For instance, an organization may have security policies and trust relationships for the whole enterprise while a number of separate administrative domains may be in existence.

This requirement can be complied with by allowing cell roots to be registered inside another cell instead of registering a cell in the global namespace. This leads into an arbitrarily nested hierarchy of cells. These hierarchies of cells with the top level cell registered in the global namespace are collected in a particular cell namespace representing an enterprise. Here we have to make the distinction between a cell which characterizes the organizational boundary, administered by an instance of a cell directory service, and the cell namespace which is the union of all cells of a single rooted hierarchy of cells.

There are a number of *benefits* of subdividing the cell namespace into a hierarchy of cells. Higher transparency and less administrative overhead in establishing the security context has already been mentioned. Creating subcells does not require the involvement of an external authority to register a cell and it does not require the availability of a global directory service for inter-cell communication within a cell name space. Furthermore, enterprise-wide central policies and administrative authorities can easily be applied to a hierarchy of cells.

Another aspect which needs to be considered by supporting cell hierarchies is the natural migration of *legacy systems* into a DCE namespace. While the creation of cell hierarchies top down may apply to newly created environments, these namespaces will usually be deployed from bottom up. Disperse subdivisions may create cells independently and want these to be connected to one uniform cell namespace eventually. Although it may disrupt the transparency goal (the distinguished names for objects changes), a facility for renaming cells must be provided for.

4. Security implications

Since the security aspects are some of the most sensitive aspects in a distributed computing environment, the naming architecture cannot be separated from the architecture of the security system. From the security perspective, a *realm* is the administrative unit for authorization control. A realm is represented by a common certification authority while a cell represents the collection of resources which belong to a common naming authority. A cell, or in a hierarchical organization of cells the top level cell, usually maps directly to a realm.

Access to entities is granted to authenticated principals (such as users, groups, and services) based on their ability to construct and present a *privilege attribute certificate* (PAC). These certificates sealed in PACs are compared with the identifiers represented in Access Control Lists (ACL).

Since it is not practical that all principals who are potentially involved in cross realm interactions establish direct links to the foreign realms, DCE adapted the Kerberos V5 model of *transitive trust* which allows access to be granted through established trust relationships between the authorization authorities.

To contain potential damage of the system, the security system in DCE imposes a hierarchical structure of transitive trust. It limits transitive trust to either those principals which share a common certification authority within a given structure or substructure of a hierarchy of cells or to those entities which directly share inter-realm keys through their certification authorities.

5. Management implications

The naming architecture in a distributed environment has to take into account that not only a very large number of nodes and entities needs to be manageable but also that the complexity of the system grows exponentially by introducing diverse technologies which have distinct and sometimes contradicting requirements on naming. Not only must the naming architecture provide for facilities which can be exploited by the management systems, but it also must be carefully designed to avoid additional unnecessary complexity.

The *scalability* is doubtlessly one of the major concerns. As discussed earlier, single naming systems in the order of 10^5 and 10^6 nodes are already in active use and the conceivable scope of addressable entities will be much higher in the near future.

One other, often underestimated, aspect of scalability is the capability of *down scaling*. Distributed system environments involving just a few systems may very well exist. Apart from possible performance overheads, system administrators in those environments must not be burdened with management functions which are only necessary in large systems. But since systems are dynamic and may grow into larger more complex environments, the same uniform interfaces must be applicable.

DCE addresses these administrative problems by providing rules for organizing the namespaces around cells, the basic administrative units and by potentially subdividing cells into smaller administrative regions (hierarchical cells). In addition, there are underlying rudimentary operations which will be specified by a common name syntax, a defined set of Application Programming Interfaces, and access protocols which define cross namespace operations through junctions.

To reduce the demand on system managers and consumers, the required security services must provide as much transparency as possible. Various protection levels must not require completely different activities. Security will not be enforced if there is no coherency, if the model is not intuitive, and if the administrative overhead is disruptive.

This requires a well balanced model which provides the basic set of specified rules and operations but also the right amount of freedom to apply organizational or personal policies. Again, cells are the domains in which policies are being defined.

However, generic administrative tasks such as user management or resource management can sensibly be performed using the technologies associated with the DCE naming architecture. Obviously undesirable behaviour, circularities and duplications such as several user accounts for the same person or different views of the directory structure dependent on the workstation are prevented by the architecture.

Other technologies layered on top of DCE such as the Distributed Management Environment (OSF DME) can further exploit the underlying services to provide this uniform and coherent system view.

6. DCE name syntax

Since DCE naming deals with composite namespaces and service implementations thereof, a number of distinct semantical rules and syntaxes must be accommodated. Some basic guidelines for the syntax of representing names applicable to the entire global namespace must be followed. These rules provide for a uniform notation of human-readable, character-string-based names. Special rules apply to the global part of a name, the global typed name syntax, describing an abstract data type for representing names and its matching rules for equality [8].

The name syntax is intended to represent unambiguously names which are hierarchically ordered as an ordered set of components. This ordering follows the rules left-to-right and top-to-bottom, where top is the superordinate finally representing the root of the tree structure.

The components of a name are separated by slashes (/). A name may have a random number of components which in turn may represent simple names or a complex structure according to the semantics of respective name system or resource manager. The pathname prefix '/...' determines the global root, while the prefix '/.:' is the signature of the root for the local cell (shortform of the global part of the pathname).

For X.500 based global names, equal signs concatenate the type value pair of AVAs (attribute-value-assertions) and comma-separated fields represent multiple AVAs. The global part of the name must not contain empty components. For example,

/.../C=US/O=OSF/OU=DEV,L=CAMBRIDGE/

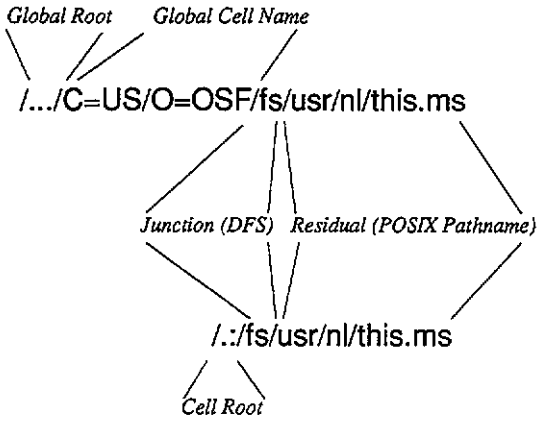

Figure 5. Pathname components.

is the name of the OSF development cell in Cambridge.

In a global namespace using the Internet DNS, the complete Internet cell name would be represented in one component in the usual dot-separated notation in little-endian format. For example

/.../osf.org/

would name the cell OSF.

The name syntax can be equally applied to typed names such as X.500 as well as to untyped names such as the DCE Cell Directory Service. A resulting global name representation therefore can be a combination of concatenated typed and untyped names.

An example of a global name for an object (file) in the Distributed File Service would look like:

/.../C=US/O=OSF/fs/usr/nl/this.ms

or equivalently, the same file would be addressed relative to the cell as:

/.:/fs/usr/nl/this.ms

Figure 5 shows the distinct components of the pathname in the example above.

One caveat in determining generic syntax rules for naming is the fact that name services and other resource managers use different character sets, need various control characters, and support different *encodings* for the representation of characters. Also, the support of *national languages* requires further considerations. While there has been some progress made in addressing these issues for standalone systems, it is largely unresolved and not standardized for distributed systems yet. This is one challenging area which requires attention in the future.

7. Portability and seamless programming interfaces

While this paper discusses primarily the capability of the naming architecture to provide for interoperability between namespaces and related issues, requirements of programmers for portability of their applications must

not be neglected. It is necessary to expose a set of well specified application programming interfaces which can uniformly be used for any integrated name service.

The DCE implementation supports two programming interfaces, the X/Open† Directory Service API (XDS) [9, 10] and Name Service Independent Interface (NSI).

XDS is the generic directory service interface which exposes the full set of semantics supported by directory services such as X.500. It is the primary API used for GDS and CDS based applications.

NSI is an RPC (Remote Procedure Call) interface targeted to manage and obtain server binding information. NSI provides the interface to the trading capability of CDS.

OSF is investigating a generic name service API which provides access to a common set of rudimentary name service operations which can be performed on any resource manager in composite namespaces which comply to the junction protocol†. This API would be on a low level, much less rich than XDS requires.

8. Summary

This paper discusses the underlying information model of the DCE naming architecture and addresses the issues involved with incorporating the various distinct needs and solutions for object naming in the distributed computing environment. It points out that the architecture provides a sound framework and that the implementation of the DCE technology has already laid the groundwork to solve naming in distributed systems.

The paper did not address application usages of name or directory services. It is obvious that the DCE naming technologies are useful for other purposes such as storing arbitrary directory information. This would have exceeded the scope of this discussion, but opens a wide field of opportunities. Also, other related technologies such as DCE Time Service and Remote Procedure Call have not been elaborated in this paper, but they should be considered as prerequisites for building the described system.

There are a number of areas which require further efforts to refine the architecture and implementations thereof. Some of these issues are subjects of further research in the community, still need to mature and may need additional standardization efforts. Areas which need most attention are:

- Providing an uniform access model (interfaces and protocols) to explore fully the capabilities of composite namespaces.
- Expanding the trading capabilities to incorporate object-oriented models.
- Finer granularity and even more flexibility in the structural organization of cells by completely solving the modelling of hierarchical cells and the associated security issues.
- Improving the management capabilities of the distributed system by integrating DME.

Finally, I would like to point out that the DCE architecture represents a synergy of requirements to solve the inherent issues in distributed systems with matured and proven technologies. One major effort taken by OSF was the integration of various orthogonal technologies into one coherent and expandable model. The first release of DCE has been shipped in January 1992. Already a number of companies have adopted DCE and its model and gained expertise with DCE in several projects. For example, the Distributed Management Environment, a technology currently being developed at OSF, extensively uses DCE facilities.

† X/Open is a trademark of the X/Open Company Ltd. in the UK and other countries.

† See footnote in section 2 of this paper regarding the XFNS initiative.

Acknowledgments

The DCE architecture and the accompanied technology implementation was only be possible as a unique multinational and multi-company project with hundreds of person-years of effort. I want to thank all the architects and engineers of supplier companies for their involvement and help and the experts of OSF member companies who actively participated in the Special Interest Group (DCE SIG), and in particular, Dr Walt Tuvell, DCE architect at OSF, who provided valuable review comments on this document. My special thanks go to Ann Hewitt for her editing help.

References

[1] Johnson B C 1991 A distributed computed environment framework *OSF Technical Paper* (Cambridge, MA: Open Software Foundation)

[2] Leser N 1990 An open distributed systems architecture *OSF Technical Paper* (Cambridge, MA: Open Software Foundation)

[3] Lampson B W, Schroeder M D and Birell A 1989 *A Distributed Systems Architecture for the 1990's* (Palo Alto, CA: Digital Systems Research Center)

[4] Open Software Foundation 1993 *OSF DCE Application Development Guide* Revision 1.0 (Engewood Cliffs, NJ: Prentice Hall)

[5] Open Software Foundation 1992 *OSF DCE Administration Guide* Update 1.0.1 (Cambridge, MA:Open Software Foundation)

[6] The ANSA Reference Manual 1989 Volumes A–C (Cambridge: Architecture Projects Management Limited)

[7] OSI, ISO9594 1988 *Information Processing Systems—Open Systems Interconnection—The Directory* Parts 1–5 ISO/IEC JTC 1

[8] OSF AES 1993 *AES/Distributed Computing—Directory Services* (Draft) (Cambridge, MA: Open Software Foundation)

[9] CAE/XDS 1991 *API to Directory Services (XDS)* (UK: X/Open Company Limited)

[10] CAE/XOM 1991 *OSI-Abstract-Data Manipulation API (XOM)* (Reading, UK: X/Open Company Limited)