In this paper we introduce two identification schemes largely used in the OSI world, and increasingly accepted in the broader telecommunications world: Distinguished Names and Object Identifiers. We show that the former scheme is flexible and powerful enough to satisfy the naming needs of a global worldwide network, while the latter can address some special requirements in particular cases. We propose a solution that allows the two schemes to coexist with minor changes to existing standards, in the framework of the same worldwide X.500-based directory service, thus avoiding needless proliferation of incompatible directory services.

Introduction

The importance of identification of communication objects of various nature has been felt at a very early development stage of the OSI standards: The basic reference model itself, in Reference Model Addendum number 3 on Naming and Addressing, introduces the concept of an Application Entity Title (AET), a high-level identifier for users (humans or applications) denoting a specific Application Entity (AE), the component of an application that performs communication functions. An AET needs to be mapped by an application layer directory into lower level addressing information, which is then used by the Association Control Service Element (ACSE) to establish associations. Neither the Addendum to the reference model nor the ACSE standard, though, specify what the abstract syntax of this parameter should be. In general, the problem of devising a uniform naming scheme to identify any kind of objects within OSI and in global internetworks needs to be addressed.

Catering to this clear and urgent requirement, specific naming efforts have been undertaken by ad hoc standards. In particular, the joint ISO-CCITT X.500 directory defined Distinguished Names within a general scheme for naming in OSI [DIR]; at about the same time, independently, the Registration Authority standard developed a scheme based on Object Identifiers, originally defined by Abstract Syntax Notation ONE [ASN.1], to address particular naming and registration requirements [REG].

This paper discusses a uniform solution to the problem of naming in OSI. First, the two aforementioned identification schemes for objects of an open network, Distinguished Names and Object Identifiers, are introduced; their relationship is investigated; trade-offs between them are analyzed; reasons for their coexistence are provided; then, a unifying solution is proposed, and operational guidelines are given for the construction of a uniform, worldwide naming space.

Distinguished Names and X.500

The concept of a Distinguished Name has been developed in parallel with the architecture and information framework of the X.500 directory standard [DIR].

The information framework of the directory is object-based [DIR2]. An entry in the directory is a set of attributes of an object belonging to a specific object class. One or more of the attributes of an object can be tagged as distinguished and used to identify the entry (and the corresponding object). All entries are organized hierarchically in a gigantic, worldwide tree reflecting the administrative hierarchies of the real world. Owing to this hierarchical nature, the information in the directory is also referred to as the Directory Information Tree (DIT); Figure 1 shows an example of a DIT section. Each node of the tree, not only the leaves, is a directory entry and contains attributes. In addition to distinguished naming attributes, entries can contain many other attributes describing information about the corresponding object.

The Distinguished Name (DN) of an entry is determined by its position in the tree: it is composed of the sequence of so-called Relative Distinguished Names (RDN) of all the entries encountered when traversing the tree from its (virtual) root to the node or leaf representing the entry. Each RDN is a set (order irrelevant) of one or more Attribute-Value Assertions (AVA) about the value of the distinguished attributes of the entry. Each AVA contains the attribute type and a purported attribute value. (In the following examples, types are shown as arbitrary alphanumeric codes; they are actually complex object identifiers, to be described in the following section.) An AVA evaluates to True when a directory entry has an attribute of the given type with the purported value. A DN uniquely names an entry if and only if all its AVAs evaluate to True. Each RDN is assigned by a naming authority. A naming authority can delegate its naming responsibility to another, hierarchically inferior, authority. At each level, an authority will make sure that the RDNs it assigns are not ambiguous. For example, when assigning a distinguished naming attribute to a new employee named “John Holmes,” the local naming authority (most likely a system administrator in connection with personnel) will make sure no other individual at the same level will have that name. Disambiguating mechanisms (like the use of middle names for humans) must be used in case of conflicts.

The hierarchical naming scheme guarantees uniqueness of names across a global network, and at the same time allows an unlimited number of variations. For example:

```
(CN=Alice CN-MTA CN-Andy)
```

The concept of a Distinguished Name has been developed in parallel with the architecture and information framework of the X.500 directory standard [DIR].

The information framework of the directory is object-based [DIR2]. An entry in the directory is a set of attributes of an object belonging to a specific object class. One or more of the attributes of an object can be tagged as distinguished and used to identify the entry (and the corresponding object). All entries are organized hierarchically in a gigantic, worldwide tree reflecting the administrative hierarchies of the real world. Owing to this hierarchical nature, the information in the directory is also referred to as the Directory Information Tree (DIT); Figure 1 shows an example of a DIT section. Each node of the tree, not only the leaves, is a directory entry and contains attributes. In addition to distinguished naming attributes, entries can contain many other attributes describing information about the corresponding object.

The Distinguished Name (DN) of an entry is determined by its position in the tree: it is composed of the sequence of so-called Relative Distinguished Names (RDN) of all the entries encountered when traversing the tree from its (virtual) root to the node or leaf representing the entry. Each RDN is a set (order irrelevant) of one or more Attribute-Value Assertions (AVA) about the value of the distinguished attributes of the entry. Each AVA contains the attribute type and a purported attribute value. (In the following examples, types are shown as arbitrary alphanumeric codes; they are actually complex object identifiers, to be described in the following section.) An AVA evaluates to True when a directory entry has an attribute of the given type with the purported value. A DN uniquely names an entry if and only if all its AVAs evaluate to True. Each RDN is assigned by a naming authority. A naming authority can delegate its naming responsibility to another, hierarchically inferior, authority. At each level, an authority will make sure that the RDNs it assigns are not ambiguous. For example, when assigning a distinguished naming attribute to a new employee named "John Holmes," the local naming authority (most likely a system administrator in connection with personnel) will make sure no other individual at the same level will have that name. Disambiguating mechanisms (like the use of middle names for humans) must be used in case of conflicts.

The hierarchical naming scheme guarantees uniqueness of names across a global network, and at the same time allows an unlimited number of variations. For example:

```
(CN=Alice CN-MTA CN-Andy)
```

Figure 1. Sample Directory Information Tree.
example, Table 1 shows the DNs of several objects in the DIT represented in Figure 1. In the figure, for space reasons, attribute types are represented as short mnemonic codes, like "CN" for "country name" and "ON" for "organization name"; they actually have numeric values, as explained in the next section.

Table 1. Distinguished and Relative Distinguished Names

<table>
<thead>
<tr>
<th>RDN</th>
<th>DN</th>
<th>Object Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountryName</td>
<td>CountryName = IT</td>
<td>Country</td>
</tr>
<tr>
<td>OrgName</td>
<td>OrgName = IBM</td>
<td>Organization</td>
</tr>
<tr>
<td>LocName</td>
<td>LocName = Rome</td>
<td>Location</td>
</tr>
<tr>
<td>CommonName</td>
<td>CommonName = Payroll</td>
<td>Application Entity</td>
</tr>
</tbody>
</table>

In the above example:

- The DN for the country "Italy" is simply (C = IT). "IT" being an internationally standardized code, from ISO 3166.
- The DN for IBM in Italy is (C = IT; ON = IBM)
- The DN for the particular IBM location in Rome is (C = IT; ON = IBM; LN = Rome)
- The DN of the Application Entity "Payroll" at this location is (C = IT; ON = IBM; LN = Rome; CN = Payroll)

The X.500 naming structure allows this; each of the objects in the DIT above is an entry in the directory. Naming uniqueness is accomplished by delegating the authority to construct names at each level in the hierarchy and enforcing RDN uniqueness at each level. For example, an authority representing the country Italy would assign (or at least endorse) the names of the subordinate organizations Olivetti, IBM, and Fiat, enforcing uniqueness of their DNs.

DNs are globally unique, because at any given level the uniqueness of the name component at that level is guaranteed by the responsible naming authority; they are flexible, because the individual components have to comply only with syntaxes dictated by requirements specific to the responsible naming authority; and they are open-ended, because the components are determined level-by-level in a decentralized fashion, up to any depth. They provide the flexibility and the power to name any entity anywhere in the world, at the cost of having a potentially large number of components.

The flexibility and power of the DN naming scheme is such that other standards have chosen to name their objects with DNs; manufacturers or users themselves may take a similar decision for their own objects. This does not necessarily mean though that all the objects named by DNs have to be directory objects, contained in the Directory Information Base. The directory services described by the X.511 standard [DJR3], in fact, may apply only to the upper portion of the DN tree, which plays the crucial role of interconnector of naming spaces. Existing naming spaces such as those used for file systems or TCP/IP mailboxes, or name spaces from other standards, can be hooked to the global tree without requiring that each object they name become a directory object. Network Management, for example, uses DNs to name managed objects [MGMT]. The only necessary step to hook a foreign name space to the DIT is that a directory object be created as some leaf of the DIT, to represent the root of the foreign name space's subtree. The DN resulting for a non-directory object will be made up of a portion contained in the directory, up to the sub-tree root, representing an object manager at a junction point, and a portion reflecting the foreign naming space. Even un-typed names can be inserted in this scheme by imposing a fixed type to each of their component names.

Object Identifiers

To understand DNs it is crucial to note that name components are typed: each AVA contains an attribute type followed by the value. The insertion of the types in the name components accomplishes a level of power and flexibility unmatched by other naming schemes: particular matching rules and character sets may be encompassed, which is very important to support natural languages, and parsing of a name and identification of its components are facilitated. Types are assigned within the scope of the particular standards where the attributes and their syntaxes are defined and follow the special syntax of Object Identifiers (OI) [ASN.1].

DNs are sequences of Integers representing names of another global tree, the Registration Authority tree; this tree, different and a priori unrelated to the DIT, is designed to allow independent identification by Registration Authorities throughout the world of the different standards they control and object types within those standards. Countries, organizations, standard bodies, PTTs, appear as nodes in the tree in their capacity of Registration Authorities; standard-specific objects like countries (as object classes) and country names, persons and common names, telephone numbers, etc., appear in the tree as well. New entries in the tree can be registered in a decentralized fashion according to specific procedure defined in the ISO 9834 standard [REG]. It is important to observe that the OI tree is not a directory tree, i.e., its nodes do not correspond to physical entries storing information (attributes) of the objects. It merely reflects a convenient mechanism for assigning world-unique identifiers to standard-related objects.

![Figure 2. Sample Registration Authority Tree.](image-url)
Figure 2 shows an example of a small part of the OI tree. In this example, the OI for the “nhs-molis” standard (X.400) is (2 6); in the context of the directory standard (2 5), the OI for the “country name” attribute type is (2 5 4 6), which is the actual value of the attribute type used in Figure 1 and in Table 1.  

In object-oriented systems the concept of a class of objects is distinct from that of an instance of such an object. Classes are typically defined by Registration Authorities as they introduce new standards and new types of objects within those. Then, when vendor organizations decide to implement those standards, they instantiate a number of those objects of the defined types. Finally, customers and end users finalize the object definition and instantiation by installing them at specific locations in their networks. At this point the “position” of those object instances in the world does not depend on their registration, but on their administrative situation: an X.400 Message Transfer Agent (MTA), for example, may be located with a particular company in some country, even though the concept of MTA was originally defined by ISO and CCITT within the X.400 standard. Such object instances are then named by DNs.

In contrast to the above interpretation, some OSI standard developers, inspired by the tree-shaped Registration Authority tree, which allows identification with such an attractively simple syntax, are promoting a proposal for Short Form Names, whereby OIs would be used in addition to DNs, transparently to humans, to identify Individual object instances rather than just classes or types. For example, file storages of FTAM (File Transfer, Access, and Management) have been identified by OIs [FTAM]. In the same spirit, other standards started looking with interest at the OI scheme to name some of their own objects. The Virtual Terminal standard [VT] introduces the concept of terminal profile, a collection of values of parameters characterizing a particular type of terminal, identified by an OI; in the view of VT, a terminal profile should be registered with the directory and made available to VT users via directory mechanisms.

While the proposal of short form names exploits properties that the OI syntax already has: open-ended, flexible, convenient (allows delegation of naming responsibilities), and concise, it introduces a new, unorthodox dimension in the concept of OI. The next section will evaluate the benefits of each naming approach and justify their coexistence. Then, a number of different approaches to dual naming, with their consequences with respect to directory support, will be briefly presented.

The DN vs. OI trade-off

OIs are concise, structured, and open-ended, but suffer the following major shortcomings:

- They are not user-friendly; this may not be a real problem as long as computer systems are able to ensure that no human users will be ever confronted with OIs.
- They are originally designed to represent object types, including all the authorities responsible for their definition, and not instances of such objects, normally, an instance of a particular object will be assigned a DN based on the administrative context the object belongs to (country, organization, etc...), whereas the structure of OIs reflects the Registration Authority hierarchy, which most likely does not coincide with the administrative structure.

- They are the expression of a mechanism devised to register objects in a globally unique fashion, and not to associate information with the objects they name: directory support for OIs is not easy to provide. In particular, OIs cannot be used to access the X.500 directory, whose search mechanism is keyed on DNs, and cannot perform efficiently an operation keyed on other types of structures.

DNs, on the other hand, have shortcomings too: due to the use of full-fledged OIs as attribute types, individual RDNs tend to be large; due to the flexibility of the naming scheme in reflecting real world hierarchies, DNs tend to be made up of numerous RDNs. The expressiveness and redundancy of DNs are welcome at the level of human users, where names must be friendly and mnemonic, but are not necessary for use by computer programs. The argument against the large size of DNs loses momentum, though, in view of current trends towards increasing memory size, processor power, network speed, and bandwidth.

These arguments may appear sufficient to rule out the use of OIs for naming object instances. Unfortunately, in many cases the “clear” distinction between object types and instances is not so clear: in the case of VT, for example, the terminal profile can be considered at the same time an instance of the type profile (to be named with a DN), and a type for a number of instances of terminals (to be identified by an OI). The X.500 directory standard itself is currently devising a mechanism for registration of schemas, whereby new objects of type schema would be supported by the directory to define object classes and their relationships: A schema object would be named with equal rights by a DN and an OI. Therefore, it is inappropriate to expect that the requirements for OI naming will completely disappear. A more realistic approach would be to limit OI naming to those cases where the distinction between types and instances is arguable, and when it is necessary to store information (attributes) about the named objects; appropriate directory mechanisms will have to be provided to that end. Solutions to this problem are addressed in the next section.

The unification of the naming trees

An attempt to fully merge the OI tree and the DIT would not be feasible, and even undesirable, for the reasons expressed above. Whenever for any reason the need arises, though, of associating information with OI-named objects, some sort of directory support must be devised. This is equivalent to building an OI-based directory tree, the relationship of which with the DIT must be specified to avoid confusion and duplication of functions.

There are three possible ways of achieving this goal. The first two assume a totally new “OI-Directory” tree (OI-DIT), which can either be supported by a separate directory, unrelated to the real DIT, or contain only pointers to the real DIT. The third solution is to insert OI-named entries directly into the DIT.

Two Separate Trees: Build a new directory tree, the OI-DIT, where entries are identified by integers instead of RDNs. A full path root-to-leaf constitutes an OI. This tree fully maps the OI tree, but unlike it allows storage of attributes related to the named objects. A special directory of a totally new design, keyed on OIs, allows retrieval of entries of the OI-DIT. The interrelationships between the OI-DIT and the real DIT may be kept to a minimum, but consistency problems will inevitably arise whenever the same object needs to be present in both trees; in this case the same object will not only get two different names, an OI and a DN, but also two directory entries, the consistency of which needs to be ensured. This non-elegant compromise

\[^{2}\text{The visual syntax used here to represent OI's on paper is purely arbitrary.}\]
solution, which with the term "federated naming" also encompasses other existing naming schemes, has a good level of acceptance in the standards bodies, but the mechanisms for consistency, as well as the OI-based directory, have yet to be devised.

One Tree, plus one Shadow Tree: There are still two separate trees, but the entries of the OI-DIT contain only aliases (i.e., full DNS pointing to entries of the DIT, as defined in [DIR2]) to the real entries in the DIT, whenever appropriate. The X.500 directory, which provides alias support, could be used for the OI-DIT as well, but a new mechanism to navigate over the OI tree would be required. The major drawback of this approach is its limitation to leaf entries of the OI-DIT, because in X.500 an alias entry in the DIT can only be a leaf: intermediate nodes of the OI-DIT could not correspond to any DIT entries, i.e., could not store any useful information. Moreover, performance problems may arise because of the amount of extra navigation necessary due to the processing of alias entries: whenever trying to retrieve a particular attribute using the OI, the navigation mechanism would first have to locate the OI-labelled leaf, and then start again using the DN contained therein to locate the DN-labelled leaf, the only one containing the attributes.

One Single Naming Tree: Selected entries of the DIT get alternate naming labels (RDN and OI portions). The X.500 directory is enhanced to perform navigation keyed on either type of label. This solution is preferable and elaborated further in the following.

The basic idea of this solution is to construct the DIT in such a way that the same path root-to-leaf will be traversed following either the DN or the OI labels. Only those OIs that represent specific object instances should be treated in this way and inserted into the DIT. (There is no reasonable need at the moment to insert into the tree those OIs that refer to pure object types as opposed to object instances, being the information related to those adequately addressed by the registration procedures.)

To this end, we introduce a new attribute type called Relative Object Identifier (ROI), with OI syntax (using the style of X.500 to define attributes [DIR6]).

relativeObjectIdentifiers ATTRIBUTE
WITH ATTRIBUTE SYNTAX objectIdentifierSyntax
:= {attributeType n}

The ROI groups one or more of the integers of the full OI of an object.

Having defined the ROI, parts of the OI tree can be folded into the DIT applying the procedure described in the following. In most of the cases, the OI to be folded into the DN tree will reflect the administrative, as opposed to standard/regISTRATION, structure of the world; for example, a particular VT profile or managed object will get an OI derived from that of the company or organization that owns the object, not from the standard which defined its type. In principle, though, the following tree-merging procedure can be applied to any OI of any structure, if such a need arises.

1. Select the relevant portions of the OI tree which need to be inserted into the DN tree. They will most likely refer to objects instances, not types; the information related to those objects will have to be stored in the DIT. It is assumed that the procedure of ISO 9834 is used for getting those OIs in the OI tree, which ensures their uniqueness.
2. Descend the OI tree grouping the integers until an object for which information needs to be stored in the DIT (e.g., a country, an organization, a locality, etc.) is encountered. That grouping of the original OI constitutes a ROI.
3. If there exists in the DIT an entry corresponding to that object, insert the ROI thus obtained in the entry as a naming attribute, and the related information as data attributes.
4. Else create a new directory entry, having the ROI as naming attribute, and containing the remaining data attributes.
5. Update the knowledge of the directory to reach the new entry (see later remarks). Note that the entry (existing or new) does not have to be immediately subordinate in the DIT to the one containing the previous ROI, since that is being taken care of by the update of the knowledge.
6. Continue parsing the original OI repeating steps 2-3-4 until the whole OI is traversed, i.e., until a leaf of the OI tree is reached.

See Figure 3 for an example of a DIT with alternate ROI labels.

This procedure will work with any OI, but since creation of new entries at the highest levels of the DIT is reserved to countries and international organizations, subject to substantial fees, its actual use will be limited to those OIs which can be inserted into already existing entries, i.e., those that most closely reflect the administrative structure of the world.

The details of the directory modifications necessary to support OI-based searches are beyond the scope of this paper. Here it is sufficient to say that the navigational mechanism of X.500 (see [DIR4] for details) needs minor modifications in order to allow searches keyed on ROIs: in particular, the directory logic must be able to parse the OI used as the key of a search and partially match it with an entry's ROI, and the knowledge about the directory structure must be expanded to support ROI-based references as well. (Unlike the RDN the ROI is a sequence of integers, not a set, which simplifies matching.) In addition, the directory information framework must be enhanced to support alternative naming attributes, i.e., synonyms.

As an example of the tree-merging procedure, let us suppose to have created an OI (a b c p v) for the town of Pavia in Italy, with which we want to associate some information, say, of touristic nature (each letter corresponding to an Integer, refer to Figure 3). (We purposely

![Figure 3. Merged DN-OI Tree.](image-url)
refrain from investigating here the reasons why one would want to name a town by an OI.) Applying the procedure, we group integers from the OI until we gather (a b c), corresponding to the entry for the country Italy in the DIT. Then, at Step 3 of the procedure, we realize that the DIT contains no entry for Pavia, so we have to create a new one (dotted in the figure). The new entry gets ROI=pv as its naming label, and contains the information to be stored with the entry. Applying Step 4, we also have to make sure that the directory knows about the new entry in order to be able to navigate to it in the next search.

As an example of OI-based navigation, let us suppose that a directory ReadEntry call is issued, having the OI (a b c f h m g r f g) (always refer to Figure 3) as the argument, to find the value of some particular attributes of the entry otherwise named by the DN: (C=IT; ON=IBM; LN=Rome; CN=Payroll). The directory will reach the location of the entry for C=IT by matching the (a b c) portion of the OI with the ROI=abc attribute. Then, it will navigate to the location of the entry ON=IBM by matching the (f h) portion with the ROI=fh, and so on until the leaf labelled with the ROI=rfg is matched with the last portion (r f g) of the OI. There, as usual, the attribute values can be retrieved and returned to the caller. Note that in some cases the OI-based navigation may well skip steps of the DN-based navigation (and vice versa), according to the two independent knowledge structures. However, if an object is named both by an OI and a DN, the navigation using the OI leads to the same location as the one that would be reached through the usual DN-based navigation: there is only ONE entry per object.

An attractive feature of this scheme is that OIs themselves can be written as sequences of ROIs using the DN syntax, using knowledge of the tree structure to group the integers correctly, thus completing the identification of the two naming schemes. In the example above, the OI (a b c f h m g r f g) can be written as the DN: (ROI=abc; ROI=fh; ROI=mg; ROI=rfg). Therefore, it is possible for an object to have only ONE name in two alternate syntactical forms, even though the user-friendliness of a DN thus obtained is debatable: it will still be desirable, of course, to maintain friendlier forms of DN for human use, at the cost of accepting two names for the same object; the use of the OI form can always be restricted to a computer, unexposed to humans.

Conclusions

In this paper, we have discussed two identification schemes defined for the OSI world and becoming increasingly accepted in the wider telecommunications world. We have shown that the DN scheme is sufficiently flexible and powerful to satisfy the naming needs of a global worldwide network, while the OI scheme is suitable to meet some special identification requirements in particular cases. We have also shown that the two schemes can be allowed to coexist, and both be supported by the same worldwide X.500-based directory service, given relatively minor enhancements to X.500 navigation and knowledge representation. A similar approach of alternate labelling can be adopted to support other existing identification schemes, thus avoiding needless proliferation of incompatible directory services, but for space and timing reasons we leave the development of such a general solution to a future work.

Acknowledgements

The comments of Phil Janson and Liba Svobodova were invaluable in enhancing the clarity and readability of this work.

References

(DIR) ISO 9594 1-8 / CCITT X.500-X.521 The Directory, 1989
(DIR6) ISO 9594 6 / CCITT X.520 The Directory: Selected attribute types, 1989