Naming and registration for IBM distributed systems

by S. Zatti
J. Ashfield
J. Baker
E. Miller

Today's trends toward interconnection of networks expose limitations and deficiencies of traditional identification schemes. The need arises for a uniform naming solution that can accommodate the size and heterogeneity of worldwide domains, while still remaining understandable, usable, and manageable by human users.

This paper describes a proposal to name objects and resources in distributed environments; each object or resource can be located, accessed, communicated with, operated on, managed, or secured using the same, unique name. The solution proposed here includes registration mechanisms necessary to ensure name uniqueness. The scheme is based on existing standards, mainly Open Systems Interconnection (OSI) Distinguished Names; whenever standards disagree, the preference goes to the alternative that offers the widest usage across all protocols. Clear, consistent naming guidelines are given that would enable IBM customers who have purchased IBM networking and system management products to name their resources so that their administrative processes and IBM's products and protocols can support those resources effectively. A method is suggested to encompass existing name spaces in a single, worldwide naming space, and a migration path is sketched. The interoperation of different protocols across network boundaries using the same naming constructs is shown by means of several scenarios. The naming and addressing scheme proposed here requires nothing new or different from the already defined standards, but allows interoperation among them by using a subset of each.

The networking world of computer systems is becoming increasingly complex: networks grow in size, new ones are introduced, existing ones are interconnected or merged. When many networks were originally installed, user friendliness and scalability features were not major concerns. Objects in these networks (e.g., resources, applications, or devices) were named according to different proprietary schemes. Since network interconnections were limited, incompatibilities between proprietary naming schemes were not a significant problem. But today the existing naming spaces, often limited to specific sizes and scopes, are unable to cope with the growth and the increase of user needs. The need for interconnection and interoperation is no longer satisfied by proprietary solutions, but calls for cooperative efforts and a high degree of compatibility. The incompatibility and limitations of existing naming spaces are becoming a stumbling block to information exchange and cooperative processing in a global multivendor environment. IBM is facing this challenge, magnified by the sheer num-

ber and diversity of its own network products. Standards bodies and manufacturers' consortia are addressing the problem as well, developing solutions for integrated naming schemes.

The proposal

This paper proposes a uniform, standards-based solution to the naming problem in a widely distributed computing environment. After defining the terms and stating the requirements for names used within a global network (similar to the one IBM networking solutions are reaching), two standard identification schemes for objects in an open network, Distinguished Names and Object Identifiers, are reviewed in some detail. Their relationship is investigated and trade-offs between them are analyzed. Next, a solution for global identification based on Distinguished Names is presented, together with operational guidelines that would allow IBM customers to build uniform name spaces that could be progressively interconnected with each other to cover a wide range of networks. This naming solution is consistent with existing standards, while at the same time providing additional value by allowing the same name of a given object to be used in different protocols for different purposes. Finally, some examples are presented that show how the proposed names can be effectively used in complex internetwork and interprotocol situations.

Terminology. The word *name* has a wide range of meanings. We use it here to mean any identifier used by humans to refer to objects that they seek, wish to have access to, or communicate with; in other words, a name conveys a sense of "what" we are talking about. When names of this kind are used in computer systems, they become a major element of the human–machine interface. In the computer world, names are strings of characters "bound" by some mechanism to a particular object or class of objects. To human users, the binding is often intuitively obvious from the context of use; e.g., a human expects "San Francisco" to mean the city in California. In computer systems, the binding explicitly associates attributes of the named entity with the name in some data record, such as a directory entry. Typically, one of those attributes is an address for the named entity.

Several levels of *addressing* may be hidden behind a name, each supported by a particular function. Addresses are normally not seen by human users, but in case they have to be, they can be considered names. An address conveys a sense of "where." Depending on the perspective, what is at one time treated as a name may at another time be regarded as an address. This definition eliminates the classical name–address duality that has generated much confusion in the past.

A *route* is a sequence of addresses or explicit directional instructions (such as go north, take link 7) leading from a particular location to a named object. A route specifies "how to get there from here."

This view is consistent with the one expressed by Hauzeur.[1] General concepts of naming and addressing are formally introduced and elegantly related to one another by Comer and Peterson.[2]

Requirements. In a naming scheme for a globally interconnected network, the following features are desirable:

- It should be *global*, i.e., users all over the globe can use the same name to identify a given object, independent of the location from which the name is referenced or where the named object resides. Context-dependent or local abbreviations (nicknames) may be usefully devised at the margins of the naming scheme, but they should not be part of the global naming scheme.
- It should be *uniform*, i.e., users must perceive they are dealing with the same object independent of the protocol or the operation applied on the object. (At present, the same objects sometimes get different names depending on what classes of operations are applied to them, and this is not satisfactory; for example, a particular printer may be referred to with one name when printing a file, with another name when checking if a particular job is finished, and with yet another one when querying about the level of the toner.)
- It should be *flexible*, i.e., not mandate any particular syntax or character representation, in such a way that existing practices, cultures, and naming schemes can be easily encompassed by the global scheme.
- It should be *open-ended*, i.e., able to accommodate growth and change according to practical needs. Past experience in the Grapevine and Clearinghouse systems at Xerox shows that naming schemes always outgrew their limits both in size and number of name components.[3,4]

The need for a new definition procedure for names arose whenever existing networks were merged and the administrators did not want to rename all existing objects. So the means to achieve scalability is to avoid limiting the number of components in a naming scheme.

- It should be *conformant*, i.e., comply with at least the minimum requirements common to all the various standard protocols in which it might be expected to be used.

In addition, it must be possible to organize naming responsibilities as decentralized, so that each organization level can be entitled to name its own objects. When the administrative requirements are combined with the quantity of names involved, any solutions based upon flat name spaces must be eliminated, and names must be structured. This conclusion has been generally accepted in the industry and reflected by the relevant standards. The remaining questions are: (1) what form or forms of name structure should be used, (2) if multiple forms are accepted, where should the various forms be employed, and (3) how should they be related to one another.

The next section reviews the mechanisms defined in international standards for identification and clarifies their relationship. The identification and directory mechanisms of Systems Network Architecture (SNA), and their relationships with both the Open Systems Interconnection (OSI) Reference Model and the OSI naming and directory mechanisms, are reviewed and dealt with in depth in Reference 5.

Distinguished Names, Object Identifiers, and their relationship

The importance of identifying network objects was recognized at an early developmental stage of the OSI standards by the international standards organization: The basic reference model itself, in addendum number 3 on naming and addressing, introduces the concept of Application Entity Title (AET), a high-level identifier allowing users (humans or applications) to denote a specific Application Entity (AE), the component of an application that performs communication functions.[6] 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.[7] Neither the addendum to the reference model nor the ACSE standard, however, specify what the syntax of this parameter should be. In general, the problem of devising a uniform naming scheme to identify any kind of object within OSI and in global internetworks was still to be addressed.

Catering to this clear and urgent requirement, specific naming efforts have been undertaken by individual standards. In particular, the joint International Organization for Standardization–Consultative Committee on International Telegraph and Telephone (ISO–CCITT) X.500 directory has defined Distinguished Names (DN) within a general scheme for naming in OSI. The X.500 directory service, in its role of *name server*, is explicitly designed to perform the mapping of such high-level names into lower-level presentation addresses. The same directory can be used in a more general-purpose manner to store information of any kind (in the form of *attributes*) about the objects identified by DN. The scope of DN is thus broader than that of the name-to-address mapping service specified by the reference model, since the directory supports not only application entities, but a variety of objects (it is more than just a name server). The standard introduces a number of classes to which objects in an open system belong, defines the attributes an object of a given class can have, and specifies a technique to name instances of such objects and a mechanism to support them.[8]

The importance of identifying network objects was recognized at an early developmental stage of the OSI standards.

The information framework of the directory is open-ended, and is not necessarily confined to objects of the classes specified in the standard document. Other standards, vendors, or users

can add new classes of objects to the directory, with any attributes, and can either publish them with an appropriate registration procedure (the classes will be shareable), or keep them confidential for exclusive proprietary usage. Other

This paper concentrates on the schemes that can be used in any standard protocol supporting the named object.

standards have also embraced the notion of DN, in particular OSI management[9] and X.400/MOTIS[10] (in the 1988 version). In addition, several schemes have been proposed to take advantage of DN in de facto standard communities, such as Internet.[11]

In an almost simultaneous but unrelated effort, the Registration Authority standard has developed an identification scheme based on Object Identifiers (OI), defined in the Abstract Syntax Notation One standard,[12] to address certain naming and registration requirements.[13] The next two sections review, respectively, the concepts of DN and OI.

Naming: Distinguished Names and the DN tree. We now discuss DNs and show how to organize them into a tree structure. The syntax of the DN is defined by the X.500 standard. However, the meaning of the DN and its usage are defined in a noncoordinated fashion by several other standards and manufacturers' consortia, in slightly different and subtly incompatible ways. This paper concentrates on the schemes that can be used in any standard protocol supporting the named object—in other words, on the lowest common denominator DN.

An entry in the X.500 directory is a data structure storing a set of attributes of an object, which is an instance of a specific object class. Each attribute in an entry is composed of an attribute type and one or more values.

All entries can be organized hierarchically in a global tree, referred to as the Directory Information Tree (DIT), reflecting the administrative hierarchies of the real world. The naming approach of the directory binds the names of the entries to the structure of the tree. One or more of the entry's attributes can be tagged as Distinguished, and used to identify the entry and the corresponding object in the following way: An Attribute Value Assertion (AVA) is an attribute-type value pair, asserting that the entry contains a distinguished attribute of that particular type having that particular value. The Relative Distinguished Name (RDN) of an entry is a set of AVAs (each of which is true) concerning all the distinguished attributes of such an entry.[14] The DN of an entry is the ordered sequence of all the RDNs of the entries starting from the root of the tree down to the entry itself. Consequently, an entry's DN is made up of its superior's DN concatenated with the entry's RDN.

Each directory entry is managed by a particular Directory System Agent (DSA), an application process communicating with peers by means of a standardized application-layer protocol called Directory System Protocol (DSP). Each DSA is responsible for the administration of a particular portion of the DIT and stores the entries of all the objects contained in that portion. DSAs are bound to each other in a hierarchical structure that mirrors the DIT portions they are responsible for and their relationships, the Knowledge Information Tree. Users have access to the directory via a service interface provided by a user stub called Directory User Agent (DUA), running locally on their machine. The DUA requests the services from the DSA by means of another standardized application-layer protocol, the Directory Access Protocol (DAP). When a DSA is queried about a particular object, it is given the name of the object as argument to the query. If the object is contained in the domain covered by the DSA, the information is retrieved and returned to the caller. If the object is not local, the DSA must locate the other DSA where the object is contained. It starts therefore a "navigation" procedure, passing the initial request to another DSA. Which DSA will be contacted next depends on the structure of the naming space, and can be derived directly from the name of the requested object and the knowledge information possessed by each DSA. The ultimate purpose of name navigation is to resolve

the given DN into the address of the particular DSA that contains the directory entry for that object, i.e., its attributes.

The following examples show a number of DNs, each consisting of a sequence of RDNs, separated by the "/" symbol. (For simplicity, in all our examples RDNs are composed of a single AVA.)

Some sample Distinguished Names:

```
* * * * * * * * * * * * * * * * * * * * * * * * *
* The various standards define how DNs should *
* be encoded in their protocols. However, there *
* is as yet no standard visual syntax for repre- *
* senting DNs on paper or displaying them on *
* screens. Nor, based upon the intensity with *
* which conflicting recommendations are being *
* advanced, should we expect a consensus to *
* emerge for some time. In the following exam- *
* ples, we show various DNs using a visual syn- *
* tax loosely similar to the one being proposed *
* for the Distributed Computing Environment *
* of the Open Software Foundation,[15] which is *
* strongly influenced by the graphic conven- *
* tions of the UNIX** world.[16]                *
* * * * * * * * * * * * * * * * * * * * * * * * *
```

People

IntOrg=GM/OU=CHEV/OU=ENG/CN=
J.C.Smith III
C=US/NatOrg=ABC/OU=SALES/OU=
NYC/CN=Gino Bartali

Places

C=US/NatOrg=ABC/NET=SALES/SYSID=
ATLVM1
IntOrg=XYCO/NET=SE/SYSID=RAL

Things

C=CA/NatOrg=BELL/APPL=ONLINE/FILE=
PRICES
IntOrg=XYCO/NET=SE/SYSID=
RAL/DEVICE=DISK1

A major difference between DNs and other existing structured names is that DN components contain explicitly the types, as in C=US, whereas the components of some other existing structured names, like the Internet *domain names*, consist of a single value, such as GOV or EDU. The attribute type, registered with the attribute definition, identifies the syntax and semantics of the value space from which the value was selected. (The less-obvious meanings of the attribute types appearing in the examples of this section are discussed as the examples are explained in the text.) Graphic sets, code-point assignments, and any special matching rules (e.g., causing "Smith" and "SMITH" to match as equal) are part of the value space definition. When a DN is received, these definitions are implied; the receiving system is expected to know how to honor them. However, rules and conventions do not flow with the DN, or appear in its visual syntax; they must be predefined to computer systems that need to honor them. A human user can guess reasonably well what matching rule to use, e.g., case-ignore. For RDNs with no special matching rules, a human user can perform the match by a purely graphic comparison and computer programs by a single compare instruction.

A major difference between DNs and other existing structured names is that DN components contain explicitly the types.

Each component must be unique only within the scope of the component to its left (assuming a left-to-right name parsing, with decreasing generality). Each component must be associated with a naming authority responsible for administering the component positioned immediately to its right. The rightmost component, the "leaf," identifies a named object. If the rightmost component is dropped, the result will itself be a proper DN, which names a different object. For example, if FILE=PRICES is dropped from C=CA/NatOrg=BELL/APPL=ONLINE/FILE=PRICES, the resulting DN, C=CA/NatOrg=BELL/APPL=ONLINE, identifies a particular application.

The leftmost components of a DN must always be either explicitly present, or inferable from the context. The uniqueness of the leftmost component is ensured by ISO-CCITT registration techniques described later in the section, Registration: Object Identifiers and the OI tree. This means that all DNs are rooted in ISO registries, and no matter how networks and name spaces are interconnected and rearranged, there will be no problems of name collisions. When the leftmost components are implied by the context and the sender chooses to omit them, the result is called a partial DN; for example, . . ./OU=SALES/BO=NYC/CN=Gino Bartali, is sufficient and unambiguous within the context C=US/NatOrg=ABC.

All the DNs used can be arranged to form a global tree in which the leftmost RDNs descend from a single "root" and every RDN to the right descends from the RDN immediately to its left. If all the DNs in the tree were represented by directory entries, then the enterprise DN tree would be a DIT. In practice, the DIT will contain only a subset of the enterprise DNs; other standards and implementers' groups, in fact, have decided to make use of the DN scheme to name their objects, without requiring that entries for these objects be registered in the directory. This is the case, for example, of many network management objects. The relationship between such DNs and those of objects having entries in the directory is not specified by any standard, but is clarified in the section on the single-DN, multiobject solution.

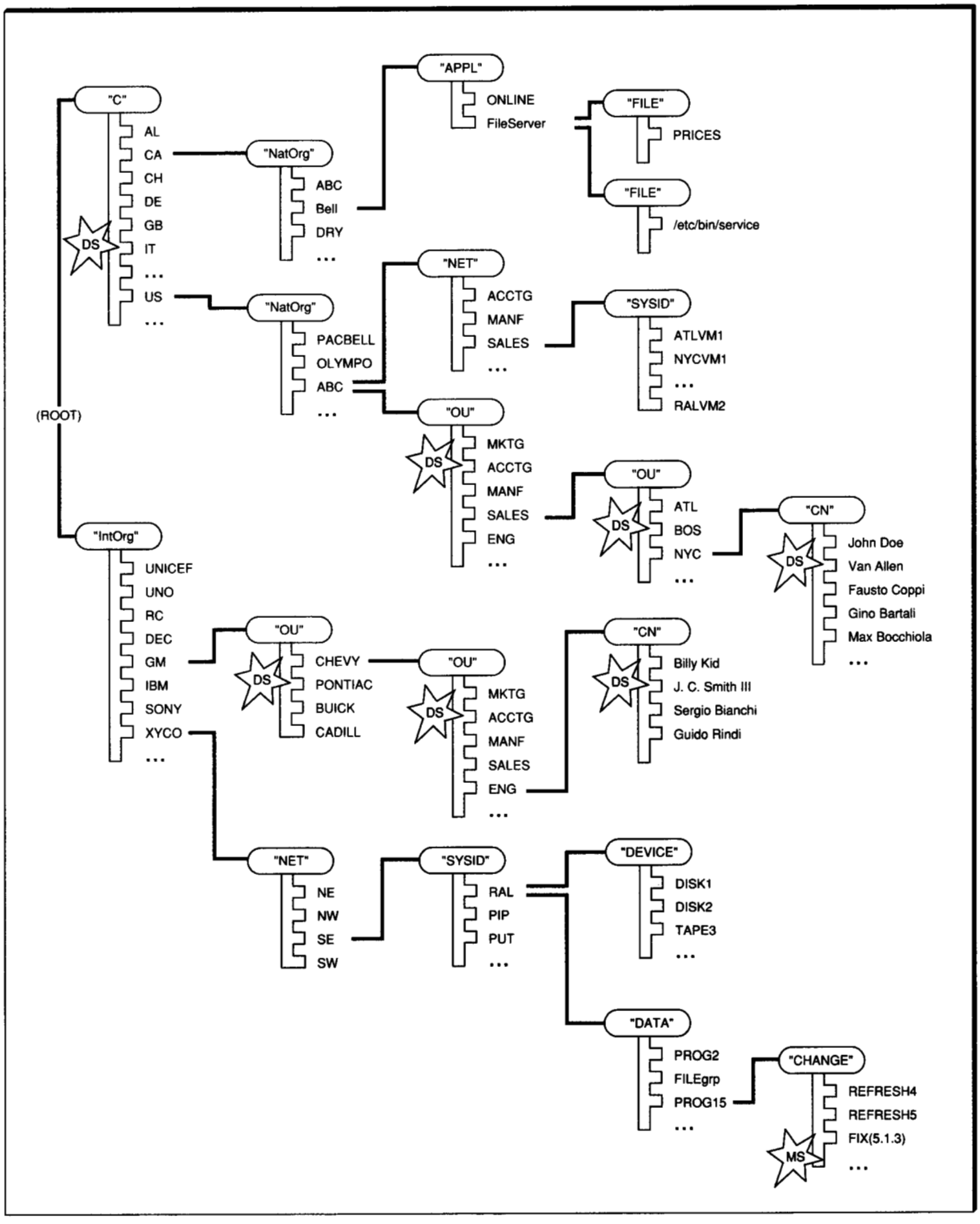
The DN tree could contain billions of DNs; Figure 1 shows a tiny fragment of the tree. Each AVA in an RDN is represented by a capsule from which a little subtree descends. The capsule contains the alphanumeric identifier of the attribute type, the thick line in the subtree represents the equality relationship that is always asserted in RDN AVAs. The thick line branches out to all the possible values that the AVA can contain. In some cases, a star-shaped symbol is associated with the trunk of the little subtree. This indicates that comparison and ordering rules beyond simple binary operations are required, as specified by the standard identified within the symbol. For example, one of the sample DNs begins with: C=US/NatOrg=ABC/OU=SALES/. . . This means that the global DN tree must include a leftmost RDN space identified by the type "C" (short for country) that contains, among others, a value "US" (alphabetic code for United States of America). The global DN tree shows this RDN immediately below and to the right of the root. Its capsule contains the "C" for country, and descending from it is the list of the two-character country codes. The DS-labeled star-shape indicates that the values for this type of RDN conform to a predefined syntax registered with the directory services standard, dictating what kind of entries can be added and how comparisons should be made. In the particular case of countries, for example, the naming value is limited to the two-character, case-insensitive country codes defined by ISO 3166.

There is no current ISO registry for international organizations, but the need is recognized.

The second RDN in the DN, C=US/NatOrg=ABC/OU=SALES/. . ., identifies the ABC corporation within some register for national organizations that is able, through an appropriate registration procedure, to ensure name uniqueness. Note that there is also an ABC corporation as a national organization in Canada; as far as naming is concerned, it must be considered a different corporation. In other words, the NatOrg=. . . RDN is unique only within a country and can only occur after a country RDN.

Leftmost RDNs do not have to necessarily identify a country: the "IntOrg" attribute identifies a hypothetical worldwide registry for international organizations that are not primarily associated with a single country. Such organizations might find it convenient to get from one registry an identifier that has no connotation of country. Otherwise, the organization would either have to choose a particular country RDN for the name of each international object it administers (and thereby risk offending people of other countries) or have multiple names for the same object (which complicates management). Currently there is no ISO registry for in-

Figure 1 An extract from the global DN tree



ternational organizations, but the need for such a registry is recognized, and while an ISO registry would be preferred, if necessary, vendor-administered registries could be enhanced to help international organizations to create country-independent DNs. The ISO registration standards and IBM's use of them is described in a following section, Registration: Object Identifiers and the OI tree.

The attribute-type and value pair structure of the RDN is flexible.

The portion of the global naming tree closest to the root tends to be fully specified by international organizations, such as ISO and CCITT, reflecting the political and administrative structure of the world. Within specific countries, naming responsibility will be delegated to specific organizations, who will build their portion of the tree according to their own policies. Other organizations, such as industry groups and vendors, including IBM, will provide certain registry services as well. Midportions of RDNs tend to be less rigidly defined, their meaning depending entirely upon their position in the DN subtree of a specific enterprise. For example, in the DN, IntOrg=GM/OU=CHEVY/OU=ENG/CN=J.C.Smith III, one OU (organizational unit) RDN is qualified by another. It is the middle portion of the DN tree where the enterprise DN designer has the greatest scope, since the root part is largely defined by standards, and the leaf part is often defined by vendors and customers.

In some cases, the shapes of the rightmost portions of DNs can be predefined into *architected subtrees*, used by services that depend upon the presence of certain information in the DN of the object being referenced. For example, in the DN, IntOrg = XYCO/NET = SE/SYSID = RAL/DATA = PROG15/CHANGE = FIX(5.1.3), the RDN CHANGE = FIX(5.1.3) is architected by IBM; IBM's change management service[17] will recognize its presence and use the value in its logic. Architected subtrees are also used in DNs that are automatically generated. For example, . . ./MACAddr = 12345/LSAPId = 12345/LSAPPairId = 12345/ThresholdControl = 5 denotes a control object within a pair of LSAPs (link-level service access points), describing the state of a particular service parameter. Millions of such LSAP pairs may exist in a large enterprise, automatically named and monitored by network management agents. Similarly, some standards specify leaf-portions of DNs for particular objects, for example, Application Entity Titles, the DNs given to Application Entities, always end with two Common Names (CN), an Application Process CN and an Application Entity CN.

The enterprise name administrators would exercise their freedom by designing the midportion of the DN subtree administered by their enterprise (possibly using RDNs such as Location=SouthEast/LocalAreaNet=Atlanta, or NatOrg=PacBell/OU=Billing), and grafting to it the architected or standard-defined subtree. Management programs would generate the DNs by automatically determining or assigning the values of the RDNs in the architected subtree of management services for each of the millions of control objects in the network. Management services would also be the primary, perhaps the only, user of those DNs. In normal operations, human users would not even need to see them.

In concluding this section on DNs, we can thus summarize their advantages: The attribute-type and value pair structure of the RDN accomplishes a potential utilization and flexibility unmatched by other naming schemes, supporting matching rules ranging from the richest variety of character sets (very important for natural language support), to the simplest, most computer-efficient and culturally-neutral naming techniques. The presence of the attribute type, in particular, facilitates the parsing of a name and the identification of its components. DNs are globally unique, since at any given level the uniqueness of the name component (RDN) at that level is guaranteed by a naming authority; they are flexible, since the individual components comply with specific syntaxes according to requirements specified by the responsible naming authorities; and they are open-ended, since the components are determined level-by-level in a decentralized fashion, at

any depth. They provide the flexibility and the ability to name any entity anywhere in the world, at the cost of having no definite limit on their length. As such, they satisfy the naming requirements expressed in the introduction.

Registration: Object Identifiers and the OI tree. To understand DNs it is crucial to note that name components are assigned types; each AVA contains an attribute type followed by the value. Types are assigned within the scope of each of the standards in which attributes and their syntaxes are defined, but no matter where they are defined, they conform to the special syntax of Object Identifiers.[18] OIs are themselves sequences of integers representing nodes of another global tree, the registration authority tree; this tree, different and unrelated to the global DN tree, is designed to allow independent identification by registration authorities (RA) throughout the world of the different standards they control and the object types (but not the objects themselves) defined within those standards. Countries, organizations, standard bodies, postal, and telephone organizations, appear as nodes in the OI tree in their capacity of RAs; standard-administered registries of entities such as country names, and standard-identified, but privately administered registries for persons and common names, telephone numbers, etc., appear in the OI tree as well. New entries in the OI tree can be registered in a decentralized fashion according to specific procedures defined in the ISO 9834 standard.[19] Tracing a path down the OI tree uniquely generates a sequence of integers that comprise a particular OI. Figure 2 shows an example of a small fraction of the OI tree. In this example, the OI for the management services (MS) standard is (2 9); in the context of the directory standard (2 5), the OI for the "countryName" attribute type is (2 5 4 6), which is what the attribute type "C" used in the examples really looks like.[18]

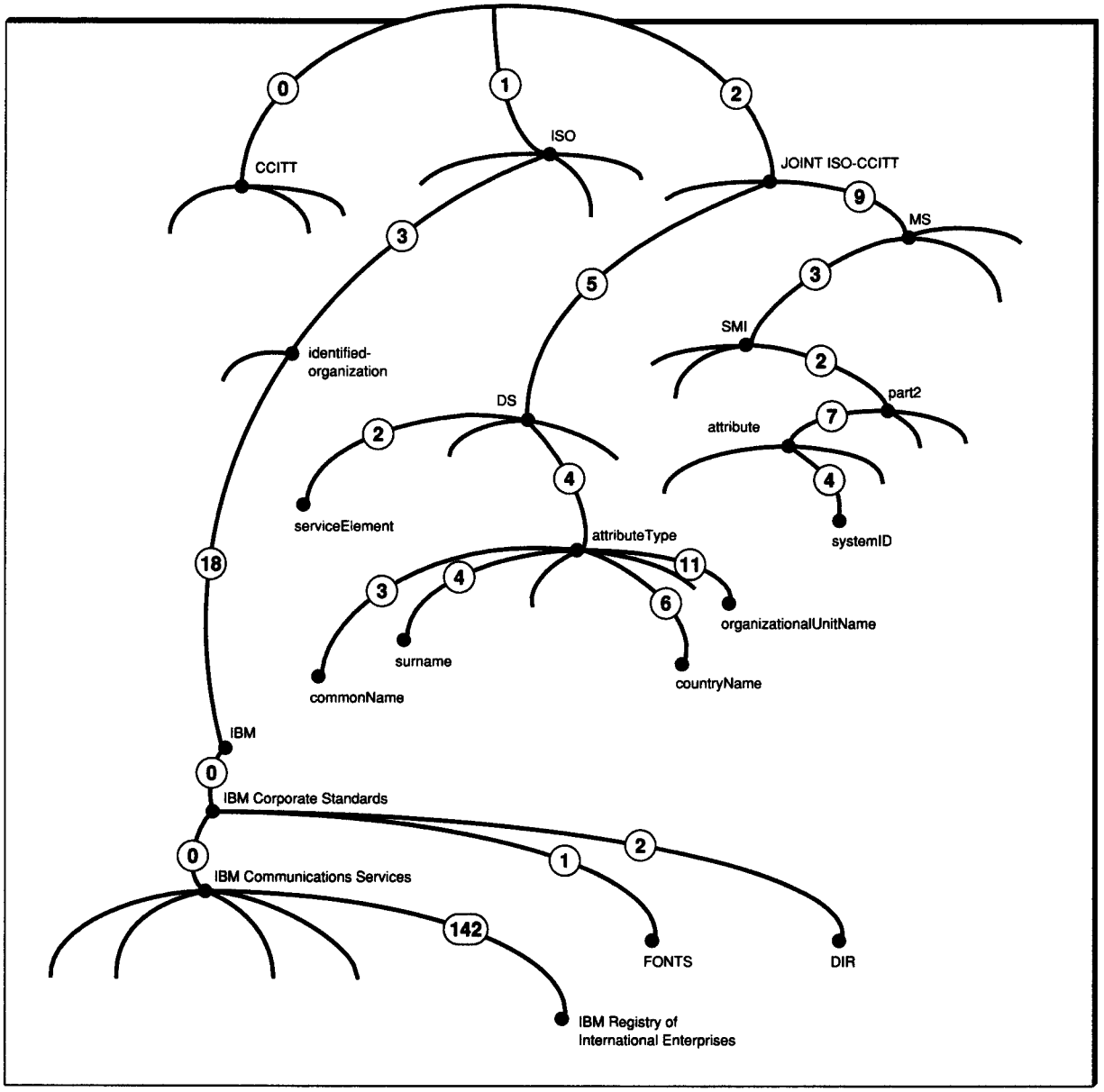
Just as in object-oriented systems the concept of class of objects is distinct from that of instance of such an object class, the concept of a standard is distinct from that of instance of such a standard. Standards are typically defined by RAs. As these introduce a new standard they define new classes of objects as part of that standard. When customers install implementations of that standard, and populate their databases, they create instances of objects of those classes. Moreover, in many cases, as systems and applications operate, they create and destroy object instances dynamically. At this point the identification of those object instances in the world does not depend on the registration of their object class definition, but on their administrative situation: an instance of an X.400 Message Transfer Agent (MTA), for example, may be located with a particular company in some country, whereas the MTA as an abstract concept was originally defined by ISO and CCITT within the X.400 standard. Such MTA *instances* are named by customer-administered DNs, whereas the definition of the *concept* of MTA is identified by a registered OI.

The uniqueness property of DNs depends upon registration and delegation.

The uniqueness property of DNs depends upon registration and delegation. Each RA is given a number by the higher-level RA that registered it. The concatenation of these numbers forms an OI uniquely identifying the authority within the world. Each RA in the tree has the power to register its own object classes and to delegate further the registration to other lower-level authorities. IBM, in particular, has become an RA with ISO, with International Code Designator 0018. As such, it is entitled to create new OI subtrees and to delegate authority to new RAs. Also, as an issuing organization as described in ISO 6523,[19] IBM can register organizations and assign codes identifying them for data interchange.

IBM is fulfilling its role as a registration authority by establishing a distributed registration structure under which its internal organizations are able to generate and administer identifiers. The registration mechanisms, when properly followed, will eliminate the possibility of the same identifier being assigned to multiple objects. This means that IBM-registered identifiers can be used in any field conforming to the OSI definition of object identifier, both inside and outside IBM. Similarly, other

Figure 2 The OI registration tree



vendor- and customer-assigned identifiers can also share those fields with no possibility of collision. The registration procedures for reviewing new requests against existing assignments, and for publishing the registry, will tend to reduce the assignment of multiple identifiers to the same object, thereby improving the extent to which two parties can share object knowledge.

In addition, IBM is fulfilling its role as an issuing organization by developing a coding system for Network Service Access Points (NSAPs), which is based on SNA Network Identifiers (NETIDs) as recorded in the SNA Network Registry. Customers who have registered their network names can use those names plus additional identifiers that they administer in creating worldwide unique NSAPs.

It should be noted that formal registration is intended for creating identifiers that will have a relatively long life and that will be used many times. Formal registration typically has some form of publication associated with it to facilitate shared use and promote interoperation. Short-lived identifiers such as transaction or connection identifiers, which survive only for the duration, respectively, of a distributed transaction or of a connection, are not registered in this sense, and their originator is not an RA.

During run-time operations, it is not necessary to decompose the OI in its internal structure.

OIs are often misunderstood as an alternative naming methodology to DNs, and in the loose sense of the word "name," which we are here trying to avoid, they may be. However, OIs serve their purpose in uniquely and compactly identifying object classes, attribute types, and various other entities used in information systems. In the context of ISO registries and the hierarchy of RAs that descend from them, the OI is the only strategic identification mechanism. In some contexts, however, OI identification has become a competitor of distinguished naming. For example, the File Transfer Access and Management standard (FTAM), uses OIs to identify *Virtual Storages*.[20] As will be seen in the subsequent section on a uniform naming solution for IBM systems, our naming scheme takes a clear position on this issue, supporting the use of OIs for registered identification of classes and class-like entities, but discouraging it for naming object instances.

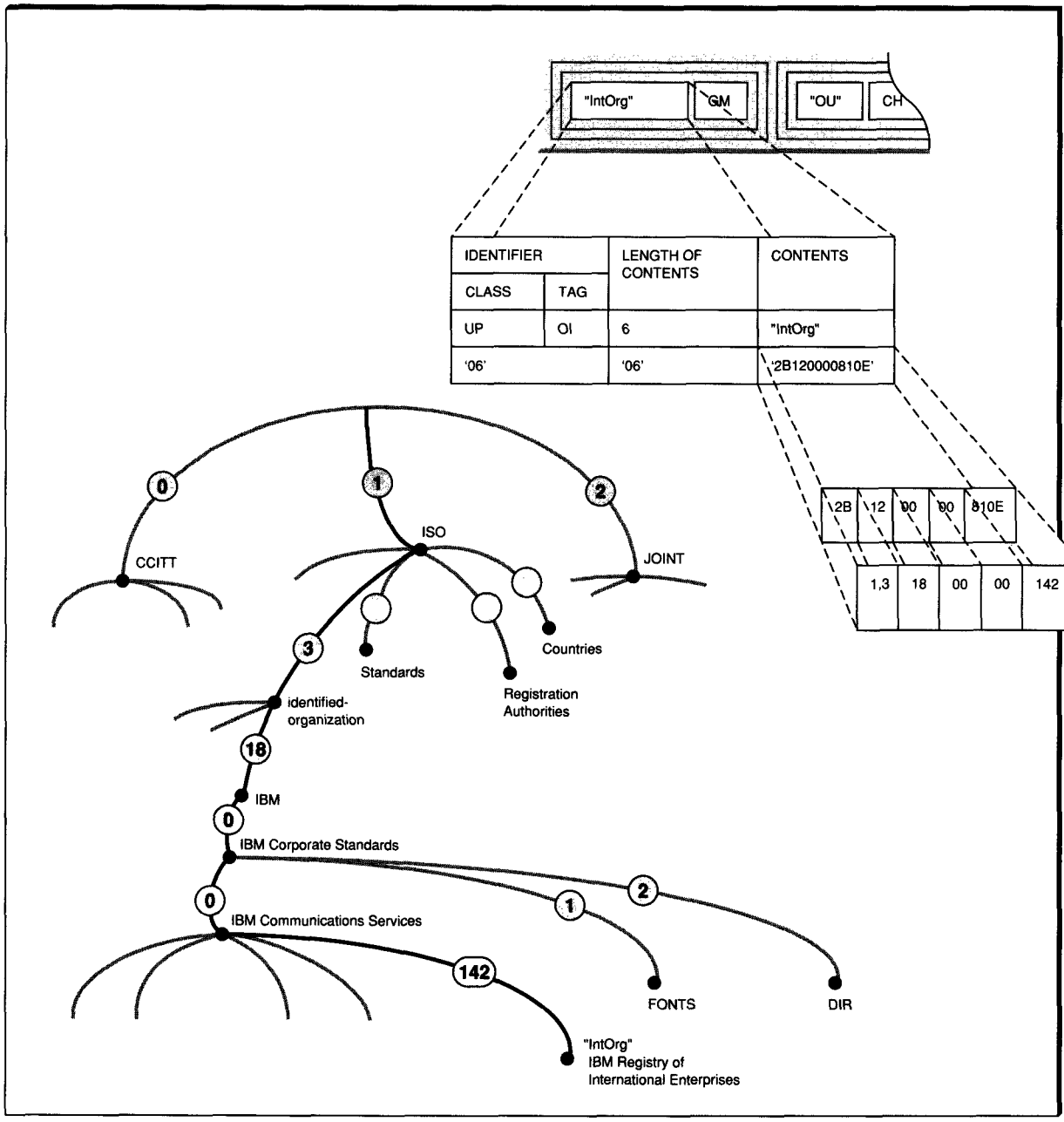
Relationship between OIs and DNs. We have seen that an OI sequence of integers defines a unique path down the OI registration tree. Each arc of the OI tree corresponds to a single numeric identifier. In contrast, an RDN contains an attribute type and an attribute value, the former being itself a complete OI.

In the visual form of the DN, the OI is usually displayed as an alphabetic abbreviation such as C for country, or OU for organizational unit. The alphabetic abbreviations are not standardized, and will probably be multidefined in the future, according to the various natural languages. Actually, when used across a network, a DN is encoded in a transfer syntax according to some negotiated encoding algorithm, most likely the basic encoding rules (BER) associated with ASN.1.[21] In the encoded form, the OI appears as a language-independent bit string where the registration information is hidden in a packed, user-unfriendly construct. Figure 3 zooms into a particular OI in a BER-encoded DN and shows how it can be decomposed into its string of integers and how its path down the OI tree can be determined. A receiving system is expected to have prior knowledge of the OIs in a DN in order to present to the users the proper alphabetic equivalent for each one of them, possibly according to the local language.

The RDN attribute values, on the other hand, do not gain or lose information when they are changed from encoded to visual form. Therefore, receiving systems do not require predefinitions of every possible value, since these values are decoded and displayed as received, but must however support the particular character set defined for the value space. This may not always be possible, especially when the receiving system is not located in the same cultural context as the sending one, as in the case of exchanges between systems in the Far East and in the Western world. Whether or not those alphanumeric values have language- or culture-specific connotations depends upon the users who assign the values.

During run-time operations, it is not necessary to decompose the OI in its internal structure; the OI is treated as a flat binary string. The internal structure of each OI serves the vital purpose of ensuring global uniqueness by capturing the chain of delegation of administrative responsibility that led to the creation of that OI, but is irrelevant to a running program. All that matters at run time is the ability to perform OI matching uniquely and efficiently.

Figure 3 Analyzing an OI in a DN



We now compare the properties of OIs and DNs. OIs are concise, structured, and open-ended, but suffer for the following major shortcomings:

- They are not user friendly; this may not be a real problem, as long as computer systems are designed to ensure that no human users will be ever confronted with OIs.
- They are intended to represent object classes, 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 hierarchy the object belongs to (such as country or organization), whereas the structure of OIs reflects the RA hierarchy, which most likely does not coincide with the administrative one.

- They are the expression of a mechanism devised to register standards and object classes in a worldwide unique fashion, and not to associate information to whatever objects they name. Directory support for OIs is therefore not easy to provide. In particular, OIs are not well-suited to efficiently access the X.500 directory, whose worldwide navigation mechanism is keyed on DNs. A possible way to fold the OI name space into the DN tree has been proposed by one of the authors of this paper.[22]

DNs, on the other hand, have shortcomings too. Since each RDN contains one attribute type (or more), and since attribute values tend to describe the object in an extensive, unshortened form, individual RDNs will be large; moreover, due to the flexibility of the naming scheme in reflecting real-world hierarchies, DNs may end up being made up of numerous RDNs. Thus DNs may be expected to be larger than untyped names, especially untyped OIs whose components are tightly-encoded integers.

In reality, the use of types may actually *reduce* the number of components in DNs, avoiding explicit mention in RDNs of names like "identified organization" or "department," so common within OIs. Thus, it is not a given that DNs will be significantly longer than other alternative names. What difference there may be is, however, of decreasing concern in view of current trends toward increasing memory size, processor power, and network bandwidth.

It can be argued, however, that while expressiveness and redundancy are welcome at the level of human users, where names must be friendly and mnemonic, alphabetics are not necessary for use by computer programs, which usually do best with integer identifiers. But if efficiency is at stake, in many cases DNs may have integer values in their RDNs, particularly in the rightmost ones, and may be thus kept short and easy to treat efficiently, just as OIs.

In addition, the presence of types confers a significant programming advantage to the DN: the structure of the hierarchical name is defined by the customer and changes from object to object, enterprise to enterprise, and day to day. The DN is essentially self-defining. If a programmed service needs to find the country code, or the architected change management subtree, it can readily do so, no matter where they appear in that particular name. An untyped name, such as the OI, is essentially positional, and thus is more constrained to design and difficult to change, and tends to require more system definition among programs that process it.

The naming scheme proposed by this paper, in conformity to OSI directory and management standards, takes a clear position in using OIs to identify classes of objects, but not the individual instances or members of a class. This makes sense when one remembers that OIs are static registered values, whereas individual instances of objects might have a very dynamic existence. It would not be appropriate to formally register a name for each one. (There is no mechanism for deregistering, so this would result in a rapid accumulation of inactive registrations.) A class definition, on the other hand, tends to have a long life and might be widely used; thus registration of its identification would be quite appropriate. The following section states the proposal in detail and justifies the choices taken.

A uniform naming solution for IBM systems

This section focuses on the IBM world and shows how the previously described naming methodologies could be used within IBM networks to identify objects and resources independent of the protocols in which the names are used. The naming and addressing scheme proposed here utilizes the already defined standards, and allows interoperation among them by using a subset of each.

The features of the DN scheme allow the same name to refer to the same physical object when used or operated upon by different standards with different protocols, thereby achieving a uniform user view. This goal corresponds to that only partially achieved in the real world by people's names, whereby the same physical person can be identified with the same name for different, unrelated purposes (e.g., registry, banking, credit, airline reservation, employment). Most likely, each service provider will make use of different attributes of the person and perform different operations on these attributes, yet accomplish this

without forcing the person to carry different names. Similarly, our scheme strives to assign to the same object—possibly used by several applications with different protocols—the same name (i.e., a DN), thus giving the user a consistent view of the various objects the user is dealing with.

The primacy principle. The bottom line of this proposal is: *DN is the primary form of naming.* Other past, present, and potential future name forms are secondary and are supported in a coexistence mode. In most cases, coexistence will be accompanied by gradual migration to the primary name form.

Experience suggests a need for a single, primary identification scheme.

DNs should be assigned to every instance of any object that needs to: have its location determined, be communicated with or accessed, be operated on and managed, be secured or administered, be moved from place to place, or be otherwise referred to in a distributed environment.

Various classes are typically defined by RAs while introducing new standards and new object classes within them, making those object classes known by registration. Then, various organizations decide to implement or use those standards, and must therefore instantiate a number of those objects. At this point the "position" of those object instances in the world will not depend on their registration, but on their administrative situation. An X.400 mail service user, an Originator/Recipient (O/R), will be located with a particular company or at a particular residential location in some country, even though the concept of O/R was originally defined by ISO-CCITT within the X.400 standard. Such instances are always named by DNs.

Some entities have both class-like and instance-like properties. For example, in SNA a transaction program (TP), an entity performing application functions and using LU 6.2 (logical unit 6.2) protocols for communication, is named with a TP name. Some TP names are registered, and copies of that TP appear at many nodes. Those TP names identify a class-like entity whose instances are the copies. Other TP names identify individual programs that exist as a single copy at one location, in other words, programs that are instance-like. So TP names are sometimes class-like, sometimes instance-like, and there is no precise test that can make the distinction. However, the fact that the class-like properties might have resulted in an OI being assigned to the entity, does not affect whether or not a DN should be assigned. If the object requires any of the functions in the above list it will also require that a DN be assigned. In the case of TPs, therefore, individual DNs should be assigned to each individual instance running on a particular system within a particular administrative domain.

It can be argued that designating one form as supreme and relegating all others to a second-class status is neither necessary nor reasonable. Considering the analogy of natural language support, we reason that all languages are theoretically equal and no one of them shall be declared supreme over all others. Extending this reasoning to the increasing heterogeneous world of computing networks, with a correspondingly increasing variety of naming mechanisms, we reason that the principle should be that no one naming mechanism should be declared supreme. All naming mechanisms should be conceptually equal, although practical constraints will force vendors to limit the variety they support. The main advantage of this approach is that it avoids offending the proponents of other naming forms and, by making no decision, eliminates the risk of making the wrong one.

But experience in other widely-distributed, interconnected, human activities, such as the following, suggests that there is a need for a single, primary identification mechanism.

- Telephony. In a process that took almost a century, telephone naming mechanisms slowly purged themselves of route connotations, and while local phone networks were progressively interconnected, the notion of a single, worldwide, telephone-number space developed.

- Languages. Although the peoples of the world are extremely proud of their own languages, at different times and different places in history particular languages were used as *lingua franca* just as means of communication. For example, Latin, Swahili, and French at different times enjoyed some form of primacy that stretched beyond the national boundaries of their native speakers. Today English plays that role and is used by an unprecedentedly wide community, mainly technical and scientific.
- Currency. In the past, currencies were often very localized, sometimes individual banks each issued its own. Commerce was much facilitated by national currencies. Now the needs of international business are pushing toward international primacy for particular currencies, extending far beyond the political influence of the countries officially using them. Examples today are the U.S. dollar and the increasingly popular ECU (European currency unit).

These "primacy" experiences are pertinent to computer network naming for the following reasons. First, the evolutionary process is the same, starting with small isolated localities and then interconnecting them into bigger and bigger groups until finally there is just one global grouping. Second, the user group is the same: the population of the planet. Market success depends upon reducing the need for computer specialists to intercede between the true end user and the network, exactly as telephone networks had to minimize the need for human operators. Third, the size of naming spaces are in the same range, i.e., 10^8 through 10^{12}. Experience with name spaces in the range of 10^2 through 10^4 (which is more typical for local operating systems) does not scale up by factors of 10^4. Individual and group idiosyncrasies, readily tolerated in small, labor-intensive environments, do not scale up to huge, highly automated environments.

The conclusion is that the basic principle of a global naming solution must be: The computer-using public must perceive one and only one name space. Anything that detracts from that perception is either not done or at least not exposed. Anything old, and already routinely used by the public, must be gracefully folded into the primary name space for the sake of uniformity and consistency.

The single-DN, multiobject solution. The X.500 directory is designed specifically to associate a directory entry, containing attributes, to an object identified by a given DN, and other standards and applications are expected to exploit this ability, by adding new attributes of any nature to the directory entry for that object. But the fact that this is possible does not necessarily mean that all the objects named by DNs must correspond to directory entries, contained in the directory information base. In many cases, actually, objects have a highly dynamic and location-dependent nature, which makes them not suitable for being represented by a directory entry, typically fairly static and location-independent. Such objects may still take advantage of the DN naming scheme, without requiring an associated entry to be stored in the directory.

From the user's viewpoint, the object is the union of all the protocol objects.

Additional naming problems arise with objects of manifold nature, that are amenable to use by different services with different protocols, each acting upon some of their different aspects and different semantics. In the strict protocol sense, there would be actually different computer objects, each reflecting a particular aspect of an object from the real world. We will use the term *user object* for these "real" objects as perceived by the human user, and the term *protocol object* for the various abstractions of this object within a specific standard or using application. If a user object is to be serviced by multiple standards, there will be multiple protocol objects defined for a single user object. From the human user's point of view, the user object is the union of all the specialized protocol objects defined for it, and possesses all of their attributes.

A global naming solution should ensure that the need for multiple protocol object definitions for the same user object does not result in the requirement for the customer to assign to it different DNs. Although a user object may belong to multiple object classes, it must have only one DN. The goal is to maintain a single naming scheme while allowing for separate object definitions. This property can be achieved by introducing an administrative procedure to be followed when instantiating and registering new objects, whereby the attributes common to all protocol objects are forced to have the same values. As a minimal requirement, the distinguished (naming) attribute must be the same for all protocol objects, including the numeric form of the OI registered for it. The procedure makes sure those attributes, and in particular those composing the name, are uniquely assigned.

Although a user object may belong to multiple object classes, it must have only one DN.

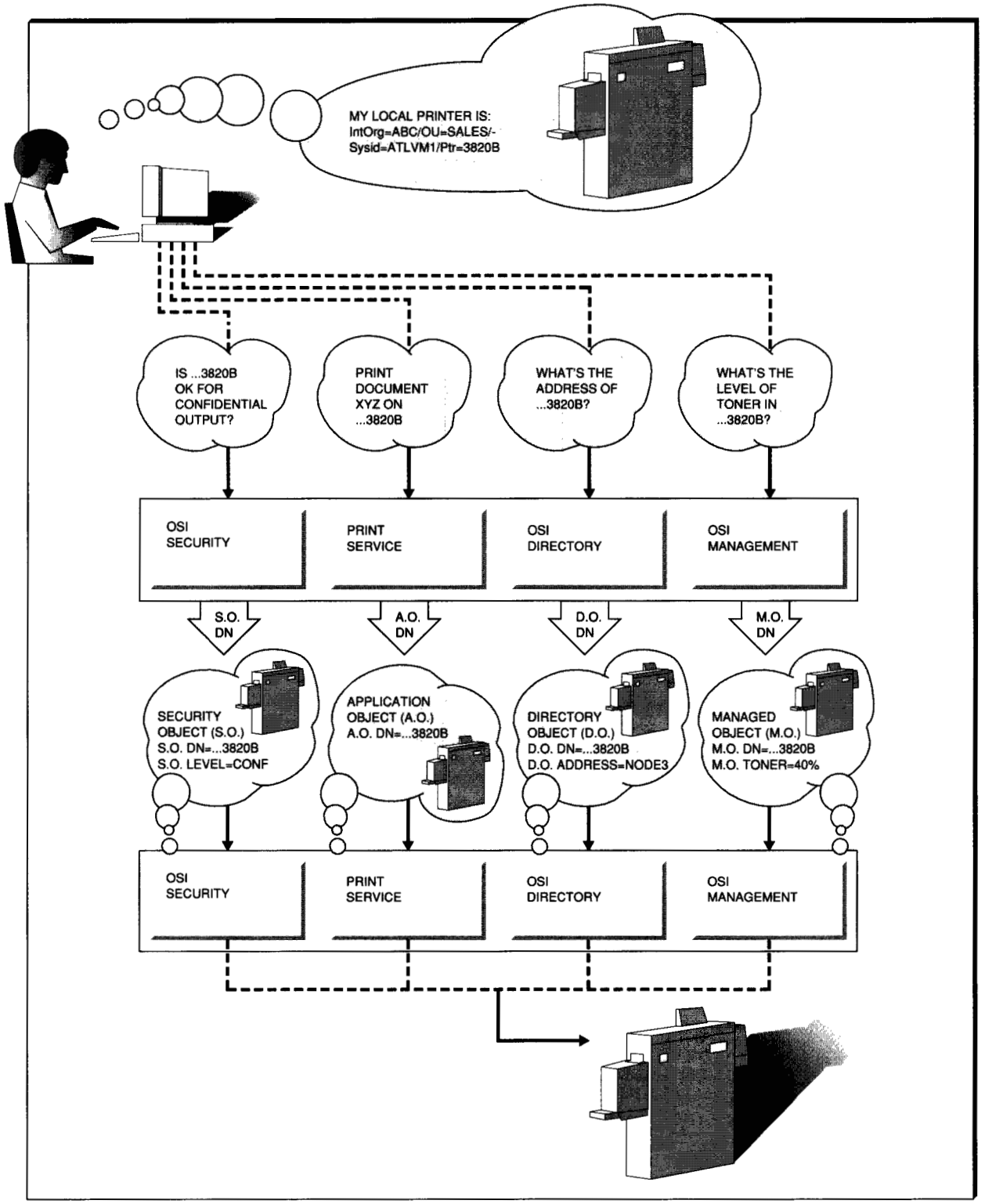
For example, OSI management, like the X.500 directory, is object oriented, and like the X.500 directory uses DNs to identify its own objects. But object classes and attributes are defined in the context of a somewhat different model. If an enterprise name designer pursued these two models independently, the result would be not only two object definitions for one resource, but two DNs. The total number of objects to be named can be estimated to be between 100 and 1000 per end user, so that a typical enterprise may have to manage many millions of names; any solution that doubles the number of those names is not only inelegant, but also extremely undesirable. Similarly, as other standards develop object definition techniques, the danger grows that they will encourage special flavors of DNs for their objects, and the administration and correlation problems will become unmanageable. The procedure sketched above avoids the problem of multiple names by providing a uniform way to instantiate new objects and assign names to them.

Figure 4 shows an example of one user object, a printer, with four different corresponding protocol objects, each used by a different service acting upon the printer. The user refers to the printer consistently using one DN. In this illustration, the user has four different reasons to refer to the printer, and each reason requires one of four different services (and corresponding protocols):

1. A security service, which the user may query to find out whether the printer is suitable for some classified output
2. A print service application, which actually sends a document to the printer for printing
3. A directory service to find the address at which the printer resides on the network (logically this would be the first service to be used)
4. A management service to determine the current level of toner

Each of these services treats the printer differently, and deals with different kinds of operations related to it. Each of them has its own definition of its view of the printer, including the DN for that object. In other words, each protocol considers that it "owns" a protocol-specific name for a protocol-specific object. From a definitional standpoint the managed-object name and the directory-object name refer to different objects with different sets of attributes. If this specialization were allowed to proceed to its natural conclusion, the managed-object DN, the directory-object DN, and all the others, would not only be distinct in definition, but possibly even in the actual values. (The illustration shows only four protocols, but there are several more that behave the same way.) However, users certainly do not want to have to know four or more names for one (from their perspective) object. The naming procedure ensures that name designers are not forced to assign multiple DNs to one object, no matter how many protocols might need to identify it. The success in reconciling the different protocol definitions depends basically upon the care the administrator exerts in avoiding any specific DN features (like multiple-AVA RDNs) that are not supported by all protocols.

Figure 4 Many protocol objects with one Distinguished Name



Unification of different naming spaces. In addition to applying this naming methodology to new objects, it is also important to be able to insert existing names, not necessarily standards-based, in the global DN tree. The DN scheme, in fact, is sufficiently open and flexible to gracefully encompass existing naming schemes, in a process of gradual migration. Whether existing naming spaces are flat or structured, they can be encompassed by the DN scheme by using either of the two techniques: encapsulation and grafting.

Encapsulation. The existing name is wholly inserted into one single RDN, with an appropriately selected attribute type. For example, as shown in Figure 1, a UNIX file name such as "/etc/bin/service", could be inserted under the DN of the file server managing it as: C=CA/NatOrg=Bell/APPL=FileServer/FILE=(/etc/bin/service). (The parentheses are used to mean that the whole file name becomes a single attribute value.) The advantage of this solution is that the existing structure is perfectly preserved and the impact on existing programs can be minimized. The components of the encapsulated name are not burdened with any extra meanings their original designers had not foreseen. The disadvantage is that the hierarchical structure of the old name ends up being totally hidden within the RDN, and is therefore unavailable to standard services for purposes such as name navigation.

Grafting. The structure of the existing name is exploded into its components, each of which becomes an RDN with appropriately selected attribute types. For example, the UNIX file system name "/etc/bin/service", inserted under the DN of the managing file server, becomes: C = CA/NatOrg = Bell/APPL = FileServer/DIR = etc/DIR=bin/FILE=service. Even names perceived to be untyped can be inserted in this scheme by imposing an appropriately registered attribute type on each of their name components. The main advantage of this solution is the elegant uniformity of name spaces it generates: all services using these names are able to exploit all components without need for any special decoding. The disadvantage is that each RDN-level name component must conform to the requirement of all pertinent standards. In particular, the resulting name must be open-ended: Shorter DNs can be obtained by dropping components such as: C = CA/NatOrg = Bell/APPL = FileServer/DIR = etc/DIR = bin, or: C = CA/NatOrg = Bell/APPL = FileServer/DIR=etc. Longer DNs can be built to further qualify objects within the scope of other objects, such as records within a file, e.g., . . ./APPL=FileServer/DIR=etc/DIR=bin/FILE=service/Record=1122. The objects thus named have certain obligations in each one of the protocols supporting them. These obligations are not particularly heavy, but they represent a change from the closed-ended, semipositional naming models that permeate many operating systems and the hearts and minds of users familiar with them. Despite the migration difficulties it entails, the DN solution has a greater potential and will be increasingly used over time.

The techniques of encapsulation or grafting can both be used for tree merging, depending on the circumstances. In both cases, though, the fundamental step is that a point in the global DN tree be selected, or created, to be the root of the subtree of the existing name space.

The subtree root, with its directory entry, represents an *object manager* responsible for all the objects contained in the subtree. In many cases, an object manager will be aware of the existence of other peer managers, and explicitly query the directory to locate them, using their own DNs, i.e., the upper portion of the object's DN. In some cases, the manager will not know this and will query the directory with the complete name. Although this query fails, the failure report does describe how much of the DN was matched, and may even, as a desirable optimization, return the address of the right manager. Otherwise, more generally, a follow-up query with the truncated DN will locate the peer manager.

The extensive usage of DNs recommended by this paper does not imply that users are compelled to constantly see and enter complete DNs. The notion of "context," omitting high-order name components when they coincide with default ones, is as normal and natural as not dialing one's area code when making local calls. Modern user interfaces can also greatly simplify the task of entering and interpreting complex constructs. Traditional nickname facilities can always be used for DNs that are frequently used, as already done today by the administrative facility of IBM's OSI/CS product, where DNs are exposed to the user only the first time they are used, and nicknames are used instead from then on.[23] In some application

areas, such as X.400 messaging, there is little variation in types, so type-suppressed displays are appropriate most of the time.

Language and culture specifics in DNs. In a worldwide internetwork spanning national and cultural boundaries, every user should be allowed to use names in the language of the user's choice. At least, the user should not be forced to use names in a language the user finds offensive. In a perfect, unilingual, unicultural enterprise this could be achieved to a considerable extent.

A global naming solution has to compromise between two philosophies.

Unilingual, unicultural situations. The limitations that arise even in this ideal environment derive from ambiguities inherent to the culture, combined with the requirements for multimedia operation and interchangeability of human and programmed operators. Humans, for example, tend to use lower and upper cases of letters interchangeably; so a human user would not react differently seeing the name "Zatti" written "ZATTI" or even (apart from aesthetic considerations) "ZAtTi." It is not the case for computers, though, since lower and upper cases correspond to different character encodings. The ambiguities are resolved by associating with each name space particular matching rules such as case-exact or case-ignore, as done by X.500.

Although the processing power of computers can take care of the ambiguities of human representation of characters, the multimedia and disconnected operations requirements state that computers will not always be present. Names may have to be exchanged on paper, by telephone, copied down from white boards, or yelled down the hallway; how can humans cope with errors and instinctive cultural assumptions? The conclusion is that special matching rules should only be used in cases where all humans who may use the name can intuitively and reliably exercise them.

When all the media are considered, unilingual, unicultural situations are almost nonexistent. A decision has to be taken on a case-by-case basis when assigning new RDNs. If the list of alternative interpretations of a particular name is small, and the context is narrow, the OSI directory philosophy of rich matching rules—equating for example Washington and WASH—works well within the appropriate linguistic and cultural boundaries; if the list of possibilities is large and the context is broad, and therefore prone to ambiguity and confusion, then the OSI management philosophy of no special matching rules works better.

A global naming solution has to compromise between these two conflicting philosophies, even in situations that are nearly perfectly unilingual and unicultural. Of all the name attributes used in the DN tree, perhaps 30 or 40, such as country, city, or common name can be expected to have humans intuitively know the appropriate matching rules. These are in most cases those defined by the directory. The remaining 100s or 1000s of name attributes should be defined with no special matching rules and used in RDNs to the right of the limited set of directory name attributes.

Multilingual, multicultural situations. Customers will require products that support naming in enterprises whose users come from different linguistic and cultural backgrounds. In such situations it is useful to distinguish between names derived from natural languages and those that are language-neutral. For example, in Canada, Ontario is a neutral name between French and English since its spelling is the same. Quebec has an accent in French, so P.Q. (Provence de Québec or Province of Quebec) is used instead. Switzerland chose the Latin name "Confoederatio Helvetica" as its own official name, to avoid privileging any one of its four languages; India chose the old European name "India" to avoid privileging any of its 14 official languages. The principle is that organizations that span linguistic groups like to use names that are as neutral as possible across all their languages. True neutrality across all the world's languages is obviously impossible, but fortunately all over the world the computer-using public has been tolerant of the Latin graphics, readily accepting names such as US, GB, ATLVM1,

London, ENG. Local, culture-specific forms of names (such as the Kanji equivalents in the Orient or Cyrillic in Russia) can always be supported by means of synonyms (inserting in the directory entry alternate naming attributes describing the object's name with different character sets), as normally happens in the real world where a Japanese person knows and uses the name both in Kanji and in Latin. Note, however, that OSI management's reports on object status do not include name synonyms or aliases, so heavy use of this solution will reduce manageability and increase definition requirements.

To allow global naming, each enterprise should strive for the level of neutrality appropriate to the mix of languages it spans. Neutrality is most needed for those resources that are referred to across the entire enterprise, and least needed for resources that are of local-only interest (assuming that everyone at that locality speaks the same language). Since the leftmost RDNs are used in the DNs of so many objects, they will usually need considerable language neutrality, in order to be successfully displayed and exchanged throughout the world. As country names are registered in a language-neutral standard (ISO 3166), so companies that want their names understood and used by many cultures should select names as language-neutral as possible.

Examples of uniform naming and usage

We now turn to showing how names can be effectively used within the enterprise and how name uniqueness can be turned to advantage by several integrated services. We do this by means of a few examples showing integrated uses of DNs by several services. No specific reference to IBM products is implied.

Distribution of services across the enterprise. Names are exchanged between the various distributed services that make use of them in basically four different ways:

1. Across interfaces within a node where the services are invoked
2. Through service-providing programs that reside at a node
3. Within collections of data residing in nodes
4. In protocol flows between the nodes that make nonlocal parts of the service available and maintain the distributed data

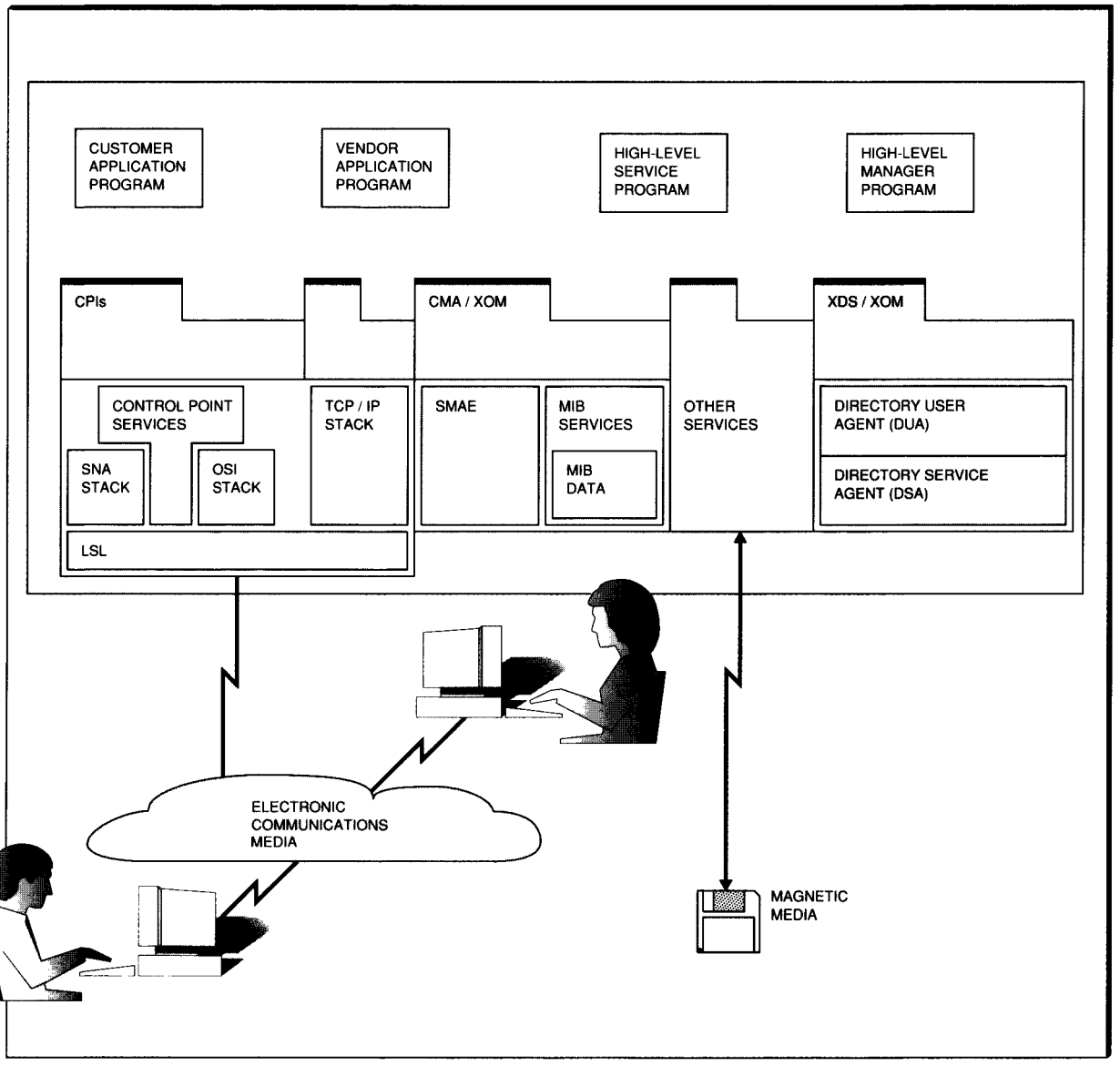
The last is rigorously defined by standards, the first not at all.

DNs have the potential of being used by a large number of services, and are stored and moved in a variety of media. Although the enterprise perceives only one DN space, the different services use DNs as if they each had an independent DN space dedicated to their particular service. Although all service protocols share electronic communications media, the only points where DNs can be exchanged between services are within nodes. Such exchanges have little or no representation in the standards, but the need remains for a local solution to allow this exchange.

On the other hand, when the DN is used outside the computer, in external media like paper, chalk board, or voice waves, if it has been constructed in the multiprotocol-compliant manner introduced in the previous section on the single-DN, multiobject solution, it has no connotation of being associated with any particular service. The notion of a single global DN tree is the natural default assumed by humans. In order to preserve that notion, the external users of the DNs must be shielded from the peculiarities of the various internal services. This shielding is much facilitated by the presence of a set of interfaces, above which the enterprise-oriented programs work with enterprise-oriented DNs, and below which each distributed service performs its speciality, interacts with the others, and cooperates in delivering a combined service to the enterprise-oriented programs and their end users.

Structure description. Traditional, vertically-oriented diagrams based on layering and the application/service dichotomy will not suffice to express the structure needed to support the enterprise DN, when applications interact with other applications exchanging DNs. The node structure shown in Figure 5 is horizontally oriented, reflecting the fact that in this situation the various distributed services interact as peers. The node is depicted containing all the key distributed services involved in naming and addressing both as service users and providers. (Not all of this function is required at every type of node.) The communication services, containing alternative stacks supporting different families of protocols (SNA, OSI, TCP/IP), are involved in all electronic communications between different nodes. How

Figure 5 Internal structure of a system



different stacks can be integrated in the same node is explained in Reference 5.

Between the enterprise-oriented applications and the distributed services is a set of interfaces. As shown in Figure 5, there is at least one interface for each service, but that need not be the way interfaces are implemented. The more services are accessed through a common interface, the better the coherence and consistency is likely to be. Notice that the top of each interface is stepped. The steps represent different levels of generality. The highest level of each interface (represented by a heavier line) supports DNs and is expected to be as system-independent as possible. The lower steps represent less-generic, more-efficient interfaces that tend to vary from system to system.

In general, either enterprise-oriented applications or service programs can use any level of interface, but normally customers write most of their enterprise-oriented applications to the high-level DN-supporting interfaces. For reasons of efficiency the services will often call each other using the lower-level interfaces.

Scenarios. The following sections illustrate how various end-user, management, and operations activities are performed within and between the various system components, using DNs to name objects on which the operations are performed. Numbered steps are used in the figures to depict the sequence of events, and correspond to labelled explanations in the accompanying text. In the figures, arrowheads indicate the direction of the caller/callee relationship, while merging an outgoing and an incoming arrow into one line indicates a call and the subsequent reply. The icons with their various shadings indicate the DNs or addresses that are being passed and returned.

Setting up an association. This example, shown in eight steps, illustrates the interactions between communication services and directory services that allow identification of remote destinations via DNs, through a high-level Common Programming Interface (CPI). (See Figure 6.)

1. The DN of the application with which the association is to be established is entered (or selected) by the end user in a visually-oriented form. The presentation services support that converts this visually-oriented form into the standard internal form is expected to be provided by the system. The user does not see any other form of name for that destination.
2. Customer application code then issues a call to the high-level Common Programming Interface for the integrated communication services requesting that an association be made with the specified remote application.
3. Somewhere below the CPI boundary a piece of control code builds and issues a directory query requesting the address attribute for the application named by the DN.
4. The DUA passes the request to the local DSA, and in this example, finds the entry locally and returns the requested address to the requester (it can be an SNA address, OSI address, or whatever else, but in a form that the destination system supports).
5. The interface code then builds a protocol-specific call (SNA LU 6.2, OSI ACSE, or whatever is available), which is routed to the appropriate communication subsystem in order to set up an association with the destination application. Although DNs are not required to identify the partners in the actual communication protocol, management policies may require that both ends be aware of each other's DNs, so that accounting records and any exception reporting are properly identified. Therefore, at least one DN must flow in the protocol during the association setup.
6. Communications services at the target end set up the appropriate control blocks and pass a handle to the partner application program (step 8). The association requires that at least one, and probably several, managed object instances be created in the Management Information Base (MIB) at both ends to represent it for management and operations purposes. Accordingly, this step introduces the required information into its local MIB and then responds to the requesting node.
7. The requesting node inserts both DNs and handles into its MIB and returns a handle to the requesting application.
8. Control is returned to the requesting application as the handle is passed to it, and the application can use the association identified by that handle for further communication.

Opening a remote file. The next example, shown in nine steps, illustrates the interactions between communications, directory, and file services to allow transparent file naming and file server location (Figure 7). Both the file server and the DSA containing its entry are remote in this case.

1. A user application program issues a file system call, FOPEN(CO=US/IntOrg=IBM/LOC=PA/DIR=a/DIR=b/FILE=c), identifying the file with a DN in internal form (a data structure in a high-level language like C, such as defined in the X/Open** interface for directory access, XDS[24]).
2. The file server does not have the file locally, nor is it aware of the structure of the file space; it must then locate another file server that has the file, using a directory call. Therefore, it issues a DS_Read(filename) to its locally available DUA.
3. The DUA accesses a remote DSA via a directory association (via the available communication

Figure 6 Setting up an association for a customer application

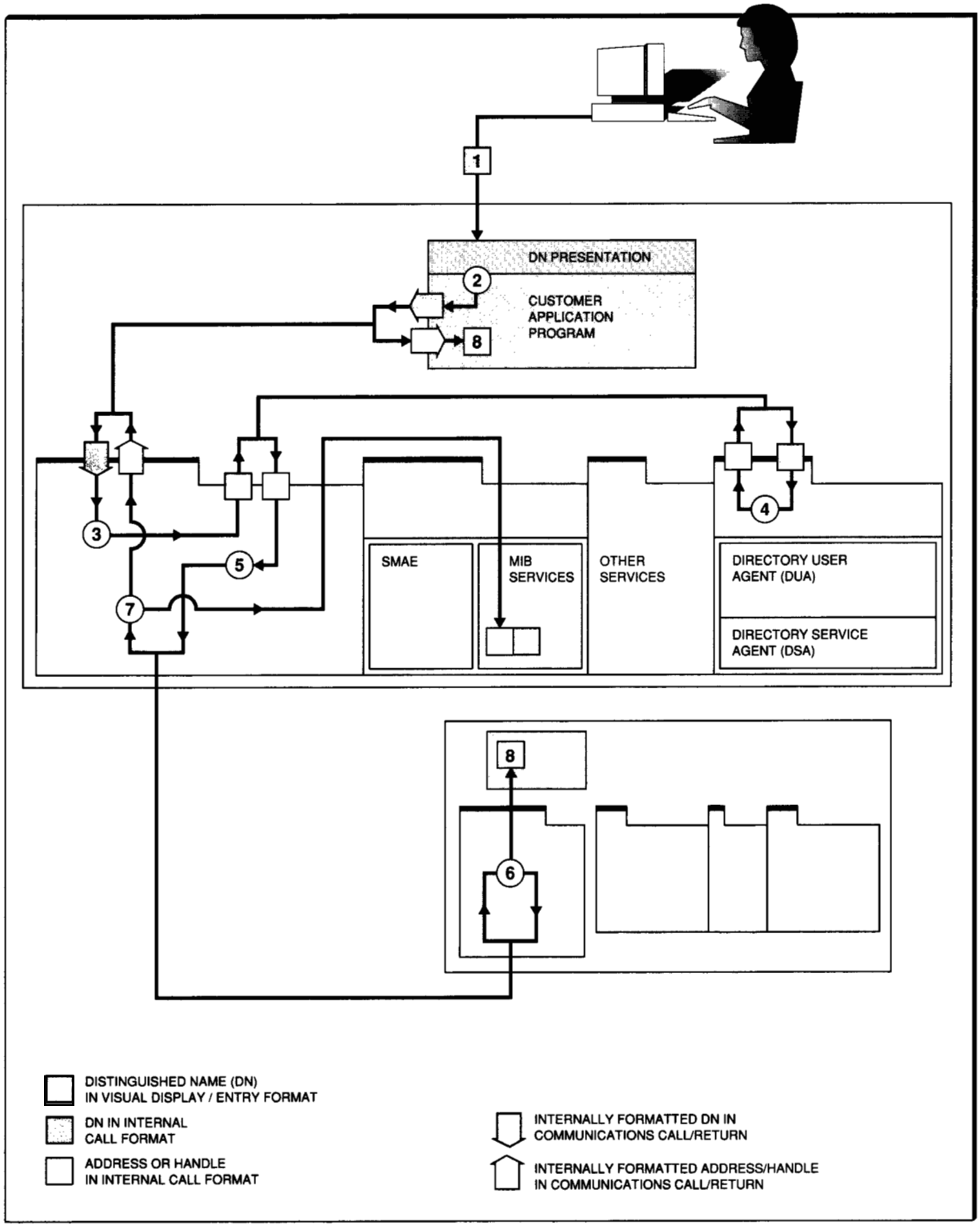
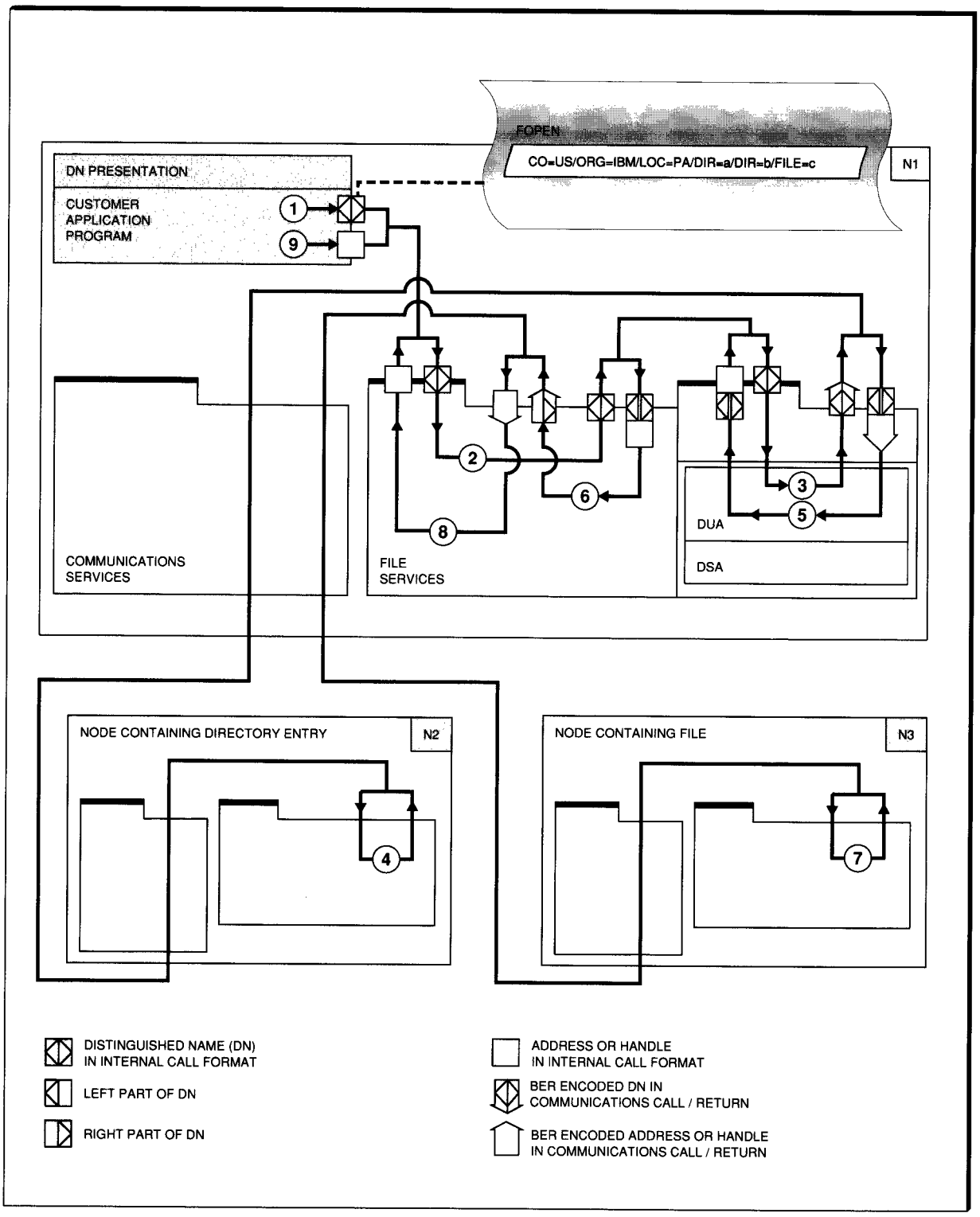


Figure 7 A file system example



protocol, whether it be OSI, SNA, or something else).

4. The remote DSA is able to resolve, through a partial match of the DN, the file name into that of the file server managing the file, and its address, which it returns to the calling DUA. (This phase may actually require a number of directory calls.)
5. The DUA at the requesting node passes the information back to file services.
6. File services uses the address of the node containing the file (N3) to send via communications services to N3 an OPENfile request with the unresolved part of the file's DN (DIR=a/DIR=b/FILE=C).
7. File services at N3 OPENs the file and returns a *handle* which can be used henceforth for other operations, on the same file-system connection.
8. File services at N1 returns the handle to the application program.
9. The application program, which had been waiting for control since step 1, now knows that the file is open and has the handle for subsequent operations. It neither knows nor cares that the file resides at a different node.

Printing a file on the most appropriate printer. This example illustrates, in twelve steps, the interactions between communications, directory, printing, and file services to allow printing of a file on a particular printer, selected on the basis of printer service features stored with the directory (Figure 8). The file server is still remote, while the DSA containing its entry is local in this case.

1. The user issues a command to print a particular file C=US/IntOrg=IBM/. . ./DIR=a/DIR=b/FILE=c on the best of a named group of printers where "best" corresponds to a computable function according to some metric.
2. Print services first calls file services, opening the remote file via an FOPEN(filename) as in example 2.

3–6. Same as example 2.

7. File services at N1 passes the handle to print services just the way it would have passed it to an application program.
8. Now that print services has the handle, and knows where the file resides, it has to locate the *best* printer. It calls directory services with a DS_Read(CO=US/. . ./devName=Printers), looking for service attributes of the printers of a particular site. Here the name refers to a whole group of printers, all the printers in a particular location. In other cases, the group name might define all printers in a domain that possess a particular service attribute (e.g., their ability to print in colors).
9. Directory services returns a list of all the printers in the group, with several attributes for each, one of which is the printer's address. In this case, directory services found the data locally.
10. Print services determines that the "best" printer is controlled by the other print services at node N3 and sends the print request there, specifying the handle and address of the file.
11. Print services at N3 accepts responsibility for the print request and returns an acknowledgment to N1.
12. When the printer becomes available, print services at N3 uses the handle and address received in step 11 to set up a session with N2 and begin the process of retrieving the file from N2 and printing it at N3.

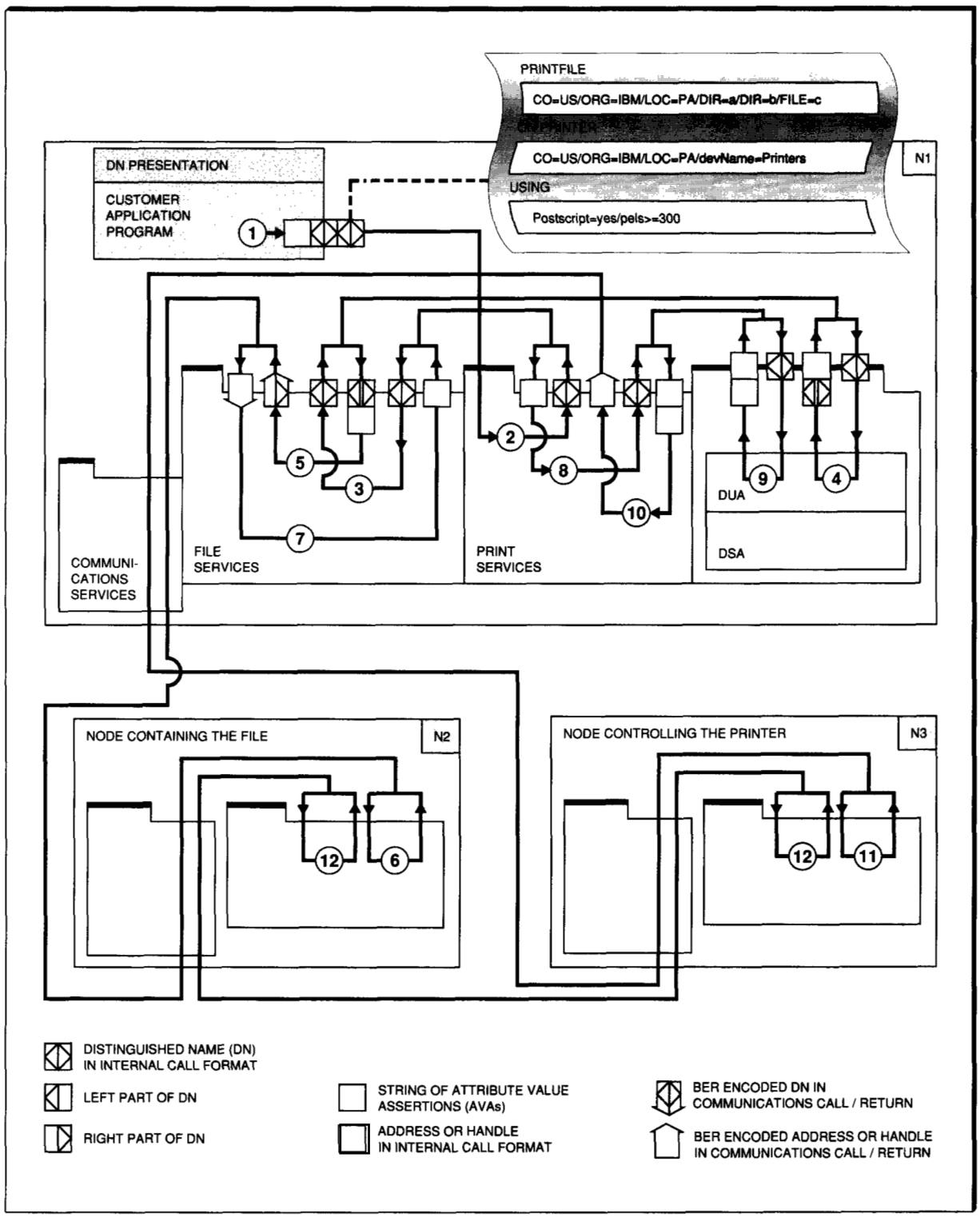
Conclusion

Today's trends toward interconnection of networks expose limitations and deficiencies of the traditional identification schemes. Mechanisms that are scalable to worldwide domains and can still be understood, used, and managed by humans are required to identify the communicating principals, authorize the use of shared or network resources, and secure the information being exchanged. This connectivity and identification requirement has been recognized by the standards community and building blocks such as Distinguished Name, Object Identifier, and OSI addressing are being developed.

The challenge for standards bodies, governments, consortia, vendors, and large enterprises is to quickly put in place a single global name tree with the underlying registration and addressing mechanisms that will simplify the administration and management of this huge and complex network that continues to grow as economics dictate and the underlying technology evolves.

The solution described in this paper is based on the use of the OSI Distinguished Name and its supporting registration and addressing mechanisms. One of the most significant advantages of

Figure 8 A file system and printer example



the OSI Distinguished Name and the supporting X.500 directory is that it will allow easy and nondisruptive migration and coexistence with existing naming, registration, and addressing mechanisms including SNA.

Acknowledgments

Roger Cheung, Mike Gering, Jim Gray, Phil Janson, Vlad Klicnick, Lucille Lee, and Liba Svobodova have greatly improved the quality and readability of the paper with their technical comments. John Hunter has provided invaluable support and encouragement.

**Trademark or registered trademark of UNIX Systems Laboratories, Inc., or X/Open Co., Ltd.

Cited references and notes

1. B. M. Hauzeur, "A Model for Naming, Addressing, and Routing," *ACM Transactions on Office Information Systems* **4**, No. 4, 293–311 (October 1986).
2. D. Comer and L. Peterson, "Understanding Naming in Distributed Systems," *Distributed Computing* **3**, 51–60 (1989).
3. A. Birrel, R. Levin, R. Needham, and M. Schroeder, "Grapevine: An Exercise in Distributed Computing," *CACM* **25**, No. 4, 260–274 (April 1982).
4. D. Oppen and Y. Dalal, "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment," *ACM TOIS* **1**, No. 3, 230–253 (July 1983).
5. P. Janson, R. Molva, and S. Zatti, "Architectural Directions for Opening IBM Networks: The Case of OSI," *IBM Systems Journal* **31**, No. 2, 313–335 (1992, this issue).
6. *Information Processing Systems, Open Systems Interconnection, Basic Reference Model, Part 3: Naming and Addressing,* ISO 7498-3, International Organization for Standardization, Geneva (1987).
7. *Information Processing Systems, Open Systems Interconnection, Service Definition for the Association Control Service Element,* ISO 8649, International Organization for Standardization, Geneva (1988).
8. *CCITT X.500-X.521, The Directory,* ISO 9594 1-8, International Organization for Standardization, Geneva (1989).
9. *Information Processing Systems, Open Systems Interconnection, Common Management Information Protocol,* ISO 9596, International Organization for Standardization, Geneva (1990).
10. *CCITT X.400 Message Handling Systems: Information Processing Systems, Text Communications, MOTIS,* ISO 10021, International Organization for Standardization, Geneva (1988).
11. S. Kille, "X.500 and Domains," RFC 1279, SRI Network Information Center (December 1991).
12. *Information Processing Systems, Open Systems Interconnection, Specification of Abstract Syntax Notation One (ASN.1),* ISO 8824, International Organization for Standardization, Geneva (1987).
13. *Information Processing Systems, Open Systems Interconnection, Procedures for the Operation of OSI Registration Authorities,* ISO 9834, International Organization for Standardization, Geneva (1990).
14. In an X.500 DN, each RDN can contain multiple AVAs, i.e., an entry can have more than one distinguished attribute. Some other standards (e.g., OSI network management) make use of DNs as well, but limit RDNs to one single AVA; if multiple-AVA DNs were specified to a management service, they would be rejected. Therefore, in some cases, multiple-AVA RDNs can generate problems in standards interoperability.
15. W. Tuvell, "Proposal for Global Typed Name Syntax," Open Software Foundation, Cambridge, MA (1991).
16. For several years the electronic mail community has used a different visual syntax (with ";" as separator) to represent X.400 Originator/Recipient (O/R) names, which have then been folded into DNs. No effort is apparent at the moment for unifying these syntaxes.
17. C. P. Ballard, L. Farfara, and B. J. Heldke, "Managing Changes in SNA Networks," *IBM Systems Journal* **28**, No. 2, 260–272 (1989).
18. The visual syntax used here to represent OIs on paper is purely arbitrary.
19. *Data Interchange, Structure for the Identification of Organizations,* ISO 6523, International Organization for Standardization, Geneva (1984).
20. *Information Processing Systems, Open Systems Interconnection, File Transfer, Access, and Management,* ISO 8571, International Organization for Standardization, Geneva (1987).
21. *Information Processing Systems, Open Systems Interconnection, Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1),* ISO 8825, International Organization for Standardization, Geneva (1987).
22. S. Zatti, "Naming in OSI: Distinguished Names or Object Identifiers?," *Proceedings of IEEE Compeuro 91,* Bologna (1991), pp. 258–262.
23. *MVS and VM OSI/Communications Subsystem, Configuration and Administration Guide*, Chapter 12, SL23-0186-00, IBM Corporation (March 1990); available through IBM branch offices.
24. *XDS Interface*, X/Open Company Ltd., Order Nr. XO/Prelim/90/030 (1990).

Accepted for publication January 21, 1992.

Stefano Zatti *IBM Research Division, Zurich Research Laboratory, CH-8803 Rueschlikon, Switzerland.* Mr. Zatti joined IBM as a research staff member in 1985 and has since been operating in the area of application services for communications, with particular interest in naming, addressing, and directory services. In 1989–1990, he was on international assignment with the IBM West Coast programming laboratory in Palo Alto, California, where he was responsible for the architecture of IBM directory services for the OSI product line. Mr. Zatti received the *Laurea* in mathematics (cum laude) from the University of Pavia, Italy, in 1980, and the M.S. degree in computer science from the University of California, Berkeley, in 1985. His research interests include operating systems, distributed systems, network security, and computer communication. He is a member of the IEEE Computer Society and a Senior Member of IEEE.

James C. Ashfield *IBM Networking Systems, 200 Silicon Drive, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Mr. Ashfield is a senior scientist/engineer. He joined IBM in 1961 and held a variety of positions in marketing for IBM Canada, Americas/Far East, and World Trade. In 1981 he transferred to architecture development. Mr. Ashfield holds a B.Eng. (electrical) from McGill University and an M.B.A. from the University of Western Ontario.

James Baker *IBM Networking Systems, 200 Silicon Drive, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Mr. Baker joined IBM at the Federal Systems Division, in Bethesda, Maryland, with initial assignments in software development under contract to various federal agencies. Later, in Poughkeepsie, Mr. Baker worked in various assignments planning and designing transportation-industry-oriented products. Mr. Baker has worked on communications systems projects since 1979 in Raleigh, including development of NCP software and performance analysis on OSI products. He has also worked on the development and use of SNAP-SHOT in capacity planning for customers' network studies. Mr. Baker received a bachelor's degree in mathematics from the University of Delaware, a master's degree in operations research from American University, and attended IBM Systems Research Institute in 1971. He is currently an advisory programmer in communications standards development, working on upper layer issues in OSI.

Ellis L. Miller *IBM Networking Systems, 200 Silicon Drive, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Mr. Miller joined IBM in 1964 at Huntsville, Alabama, as a member of the FSD team responsible for the development, assembly, and test of the NASA Saturn Apollo Launch Vehicle Instrument (IU). During this time he participated in the testing and launch operations of the guidance and control system of the Saturn Apollo Launch Vehicle. He later worked on diverse FSD projects including performance monitoring systems for diesel electric locomotives and in the modernization of the B52 bombing and navigation systems for which he received an IBM Outstanding Contribution Award. In 1978, he joined the SNA architecture group where he received IBM Invention and Outstanding Contribution Awards for his work on SNA network interconnection. Mr. Miller is currently technical staff to John Hunter, Director of Architecture and Telecommunication, with multivendor connectivity and OSI as one of his major areas of interest.

Reprint Order No. G321-5478.