Integrating Multiple Directory Services

Margaret Olson
Laura E. Holly
Colin Strutt

The Integrated Directory Services (IDS) infrastructure implements a directory-service-independent interface. The IDS infrastructure is used by applications that store and retrieve information about resources in environments with either multiple directory services or one of several directory services. The IDS interface isolates users and application writers from the unique requirements of different directory services by providing a view of a single, logical directory service through a simple federation mechanism. To retrieve resources from the logical directory, IDS determines its physical location and converts the resource from a directory-specific to a canonical format. Extensible schema tables represent the canonical format for each resource and allow IDS to represent resources created using both the IDS interfaces and the directory-specific interfaces.

Digital has developed the Integrated Directory Services (IDS) technology to provide a mechanism for integrating multiple directory services into a single system. In this paper, we examine the development of the IDS infrastructure. We begin by discussing the problems faced by network directory applications. Next we describe our design goals, the IDS infrastructure, and our initial implementation on the PATHWORKS product. We conclude with a brief discussion of plans for future development.

Directory Support in Multiple Environments

Although directory services are a powerful mechanism for distributing and accessing certain kinds of information, relatively few applications choose to use them. Digital's PATHWORKS application was in need of a directory for printers and file shares. PATHWORKS is a network operating system (NOS) integration product that gives users access to both Microsoft's LAN Manager and Novell's NetWare file and print shares. As we studied how to incorporate directory support into PATHWORKS, we came to a better understanding of the problems faced by directory applications in general.

Networks are growing rapidly, as are the amount and kind of information that can be accessed through the network. We were certain that future network application products would have an even greater need for a directory, and therefore a general solution was needed. We then set out to design a system that would remove the barriers to directory service application usage and deployment. We resolved the tension between the product deadline and the time required to implement the general solution by designing a complete solution and implementing what was necessary to prove the design and to meet the immediate needs of the PATHWORKS product.

Existing Directory Services

There are a number of general-purpose directory services. Some of the more familiar include X.500, Novell's NetWare Directory Service (NDS), the Cell Directory Service (CDS), and Banyan Systems'

StreetTalk.[1–4] In the past, directory services were in relatively limited use because most directory services were tied to either an operating system or a transport or both. In addition, directory services were connected to a multitude of application programming interfaces (APIs) that were incompatible and difficult to use. More recently, directory services have been tied to network operating systems or applications, rather than to host operating systems or transports. If anything, the number of "standard" APIs has grown.

In large networks, this complexity has resulted in the proliferation of directories, often containing overlapping information. This makes the network manager's job difficult, which in turn creates resistance to directory applications. At the same time, network and NOS technology has developed to a point where an ever-increasing amount of information is being shared on different machines. To give a simple example, almost every server at Digital's Littleton site has a connection to the high-volume printer in the copy center, with a different name on every server. A directory would simplify users' access to this single physical resource by presenting a single name for the printer, if only the application writer could figure out which directory service to use and how to use it.

Other Approaches

As discussed later in the Design of the IDS Framework and Service Providers section, IDS defines both an API and a service provider interface. Support for any directory service can be provided by writing a service provider module. Microsoft's OLE Directory Services (OLE DS) takes a similar approach to IDS, with a more limited initial implementation.[5] Although the current IDS implementation runs under Microsoft Windows, it was designed to port to other systems. OLE DS depends on features of the Windows operating systems.

The X/Open Federated Naming (XFN) specification was not complete at the time we were designing IDS, and it did not include either a service provider interface or a reference implementation.[6] We did examine the XFN draft and designed the IDS interface to be compatible with XFN, with a view toward supporting the XFN API in the future. Supporting the XFN interfaces on top of IDS would be a relatively straightforward task, and we have considered doing this.

The PATHWORKS Application

In the NOS environment, each NOS has its own directory or pseudo-directory. NetWare version 3 implements the Bindery; NetWare 4 implements NDS.[7] The various implementations of Microsoft's LAN Manager protocols provide a virtual directory based on information maintained by its domain controllers. In a multiple NOS environment, the user is presented with multiple information sources from the multiple directories. Even worse, the user may be faced with multiple information sources even in a single NOS environment, since there may be multiple NetWare Binderies or LAN Manager domains.

Multiple NOS environments do not, in and of themselves, cause complexity and confusion. Problems arise when people within a single environment want to share resources across multiple environments. For example, consider a common local area network (LAN) configuration where NetWare is installed on the clients and servers for one department and Microsoft's LAN Manager (contained within products such as Microsoft's Windows for Workgroups, Windows 95, and Windows NT operating systems, or the LAN Server product from International Business Machines Corporation) is installed on the clients and servers for another department. If each department's resources, users, and administration personnel are kept distinct, there is no problem. However, any desire to allow users to share resources between departments, or to have common administration over the departments introduces administrative and user problems. If a printer is to be shared by the two departments, it must be administered twice: once in the NetWare environment and once in the LAN Manager environment. Users in the two departments use different names for the same printer. Later NOS implementations, such as Digital's PATHWORKS version 5.0 or the networking software built into Microsoft's Windows 95 that provides support for multiple NOS protocols, do nothing to manage the multiplicity of names for the same network resource.

As we were contemplating the set of capabilities we needed to design for the next generation of PATHWORKS client products, we realized that solving the connectivity problem implied in a multiple NOS environment was not enough. User access and administrator control of NOS resources needed to be considerably simpler.

As we looked at the problems in larger networks, we saw the need for the ability to provide more sophisticated means to locate NOS resources. Typically, NOS client software provides the means to browse the network to locate a resource. However, browsing requires the user to know the location of the resource, specifically the name of the server, and to be able to choose the resource on the server by recognizing something about the resource name or a resource description provided by the administrator. What was needed was a design that allows a user to search, as well as browse, for a resource based on various attributes describing the resource.

Finally, existing NOS environments have a fairly limited view of the set of resources that can be referenced.

Both NetWare and various LAN Manager implementations provide support for printers and file shares. We wanted to be able to extend the types of resources that could be referenced and managed from the new directory capability that we were designing.

Thus we embarked on a design for the facility we initially called IDS, for Integrated Directory Services. The PATHWORKS version 6.0 implementation was eventually called Directory Assistant. We refer to this technology as IDS throughout this paper.

Design Goals

As we looked at the requirements of the PATHWORKS product, we found that many of those requirements could technically be met with any directory service that was integrated into the PATHWORKS applications and tool sets. PATHWORKS required the ability to

- Give a single name to resources that can be accessed by means of multiple servers or protocols
- Insulate end users from changes in the way resources are allocated among the servers
- Manage resources in an NOS-independent manner

We could not simply pick a directory service and integrate it into PATHWORKS, because we could not require that all customers deploy a particular directory service at their site. The PATHWORKS product is both NOS- and transport-independent; introducing such a dependence was unacceptable. We quickly realized that these were the requirements that kept many other applications from using directory services.

Our assumption was that many network applications would use directory services if they could, but that few of them could assume or require a particular directory service. Working from that assumption, we selected the following design requirements for IDS:

- Directory service independence
- Ability to access existing data
- Ability to join disparate namespaces into a single, logical namespace
- Removal of barriers to successful deployment of a wide area network (WAN) directory
- Ability to hide directory name syntax
- Support of search
- Support of application-specific directory entries

Directory Service Independence

Customers must be able to choose the directory service in which they store resource information. Some customers have a preferred directory service, which they want to continue to use. Other customers, who are not using a particular directory service, prefer that Digital provides the directory service. In a few cases, a customer might wish or even need to store information about different resources in different directory services.

Ability to Access Existing Data

A great deal of information currently exists in application-specific directory services and in NOS-specific directory services. A relatively large number of applications also use the native interfaces to store information in the NOS directories. Allowing users to access this information directly through IDS was critical. We expressly wanted to avoid the need to duplicate directory information in separate, incompatible systems.

Ability to Join Disparate Namespaces into a Single, Logical Namespace

Many directory services are aimed at a specific application or a set of applications. For example, current X.500 deployments contain mostly people information such as names, phone numbers, and electronic mail addresses. (Note: X.500 is an extremely flexible directory service that can be used to store almost any kind of information, but for historical reasons most deployments contain people information.) NOS directories contain information about NOS resources such as printers. Consequently, many user environments have multiple directory services, each of which contains critical business information. To access this existing data and present it to the user in a meaningful way, these multiple directory namespaces must be joined into a single, logical namespace.

Removal of Barriers to Successful Deployment of a WAN Directory

Hierarchical directory services generally require that the naming hierarchy be designed before the directory is deployed. Since the hierarchy consists of names, and names are sensitive and political entities, this can be an extremely difficult task. Organizations also change over time, further complicating the problem of designing a name hierarchy.[8]

Organizations that successfully deploy directory services do so from the bottom up. The NOS directories are deployed precisely because they avoid the problems inherent in a name hierarchy. An administrator can set up a Novell 3.x Bindery for a local organization without worrying about how the name of one group relates to the names of all the other groups. The downside to the NOS directories is that they have a limited ability to scale beyond a LAN. With IDS, we wanted to provide a framework that would grow with the user's environment. A user could start with a local directory but incorporate that directory into an enterprise or global directory when the time was appropriate, without affecting the end users or the applications.

Ability to Hide Directory Name Syntax

The syntax of the names in hierarchical directory services varies not only from one directory service to another, but in some cases from one implementation of a single directory service to another. The syntax for Domain Name System names is ordered the same as a postal mail address, that is, from the most-specific component.[9,10] For example, a machine at Digital might be bigAlpha.digital.com. The X.500 name order is usually (depending on the implementation) the reverse. The corresponding X.500 name might be: c=us;o=Digital;cn=bigAlpha. Particularly in the X.500 case, different systems and applications also accept different separator characters.

Together, the IDS designers have much experience with a number of directory services and their name syntaxes. Users and applications developers alike have been quick to point out the problems with directory names. These names are cumbersome, confusing, or just plain inconvenient to type. The separator characters within a directory name may have special meanings on some operating systems.

Because of these limitations, we decided that a name syntax specific to IDS would detract from the value of the solution. An application using IDS may choose to present its own syntax, one that is suitable to its particular environment and preferences. The API takes the object name and the context, as described in the Contexts section. The service provider module uses these to construct the name in the native name syntax.

Support of Search

Users need to locate resources in a number of ways. The most familiar method is to locate resources by knowing their name; this is often referred to as a white pages lookup, named after the printed U.S. telephone directory of alphabetically ordered names. Searching for resources based upon information about the resources is referred to as a yellow pages lookup, named after the printed U.S. telephone directory organized by business category. To support yellow pages lookup, resources must be retrievable from the directory service based on their attributes. For a printer, this might include the type of printer, the location of the printer, whether it supports color or not, who is responsible for maintaining the printer, and other information. IDS needed to support both yellow pages and white pages lookups.

Support of Application-specific Directory Entries

We saw a need to support two kinds of extensibility: the ability for an application to create new kinds of directory entries, and the ability for a customer to add attributes or other descriptive information to the directory entries created by PATHWORKS or other applications. By providing applications with the capability to create new kinds of directory entries, the IDS designers allowed IDS to be used by any application, regardless of its requirements. By allowing the addition of attributes to existing directory entries, we allowed customers to easily add information that is specific to their organization to application objects. For example, a customer might add a specific code, such as an asset identification tag, to all printer directory entries.

Design of the IDS Framework and Service Providers

IDS is an object-based system that consists of a framework and a set of service providers. For clarity, we further divided the framework into an API and a service provider interface (SPI). The API consists of a subset of the framework's objects and their public virtual methods. The SPI is a generalized, directory-service-independent interface (described in detail later in this section). The SPI objects define the abstract interface to the directory service. We use the term *service provider* to refer to any directory service that provides IDS storage. The service providers interact with the framework through the SPI.

Framework

The framework performs three major functions:

- It specifies the IDS directory-independent operations.
- It dispatches operations to directory-specific modules for execution.
- It verifies that all IDS objects and operations do not violate the IDS schema.

Figure 1 illustrates the structure of IDS. When an application makes an API call, the framework examines the name information and calls the appropriate service provider. The service provider then makes the call to the appropriate native directory service client. When the directory client returns the results, the service provider converts the results into the IDS canonical form. The design supports junctions from one directory service to another, in that the result returned to the framework by the service provider may be only a reference to an object in another directory service.

The abstract interface to the directory service ensures that IDS provides applications with a consistent level of functionality without regard to which directory service a customer has in his or her environment.

Because the words "object" and "object class" are overloaded and overused in the industry, we define the words "resource" and "resource class" to denote objects represented in IDS. A *resource* is a directory

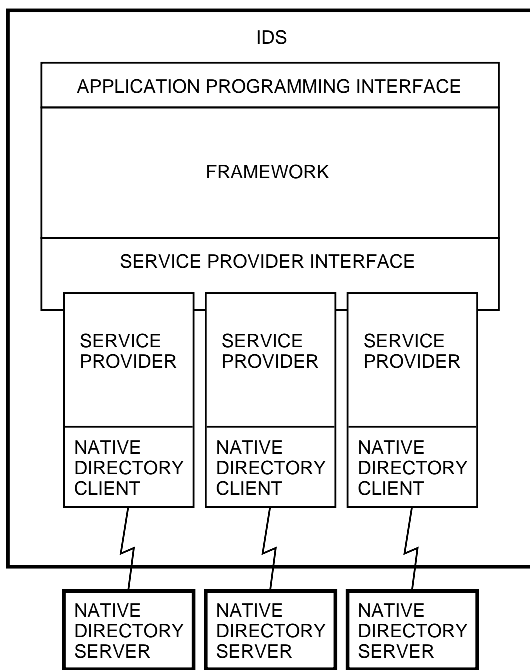


Figure 1
Structure of the Integrated Directory Services

entry; it is a directory service object that represents some network object. A *resource class* is the definition of that type of directory entry. For example, the directory entry that describes a specific printer is an IDS resource, and the IDS class that describes every printer entry is a resource class.

The framework provides extensibility by defining C++ object classes that allow for the creation and manipulation of resources, attributes, and attribute values in a type-independent manner. The type independence allows both applications and the framework itself to manipulate IDS attributes and attribute values without knowing their types. As long as the new types are built on top of existing IDS system types, application writers may define new IDS types without modifying the service providers.

The framework dispatches directory operations to the appropriate service provider and maintains overall system state and integrity. It maintains a list of the service providers that are currently available and shows the errors encountered in any failed loads. This allows the system to continue to operate, albeit in a degraded state, even though one of the service providers may be malfunctioning.

Before we discuss the design of the SPI, we describe the framework's objects.

IDS Entry The fundamental IDS object is the canonical representation of a directory entry, the IDS entry. The IDS entry is an abstract object. To create a resource class, applications define a resource type and derive it from the IDS entry. IDS entry objects are created and manipulated through the API and translated into the appropriate native directory format by the service providers. Derivatives of the IDS entry may define additional methods, but they may not override the IDS entry methods. The IDS entry methods are part of the framework.

The IDS entry methods fall into one of two categories: those which manipulate the attributes and values contained in the IDS entry in a type-independent manner, and those which perform operations on the directory. Each IDS entry, each attribute, and each attribute value contains a type. For convenience, derivatives of the IDS entry may define additional methods that manipulate certain attributes or values directly. For example, a derivation that defines a printer might define a method to set the description attribute. The implementation of this method would call the general IDS entry attribute and value manipulation method to set the value of the appropriate attribute.

As shown in Figure 2, the IDS entry contains identifying information and the attributes and attribute values that describe the resource. The context identifies the service provider that performs directory operations on this entry and the location within that directory service in which this entry is stored. The resource type defines the kind of resource that this entry represents. The resource name is the name by which applications and users refer to the entry.

The attributes of the entry are contained in a set. Each attribute in turn contains the value or list of values associated with the attribute.

Contexts The context is an object that uniquely identifies a particular location in a particular namespace. The IDS context is very similar in concept to the XFN context.[6] All contexts contain the type identifier for the directory service and an internal name. The type identifier is used by the IDS framework to dispatch operations to the appropriate service provider. The internal name is the location within the directory service described by this context. The internal name is represented in the native syntax of the underlying directory service. The service provider is responsible for setting and maintaining this internal name. (See Figure 2.)

Attributes and Attribute Values The type of an attribute defines the data type of its value or values. The attribute value object is a canonical representation of an actual attribute value. The attribute value object defines a set of methods for accessing and manipulating values. For each data type supported in IDS, there is a corresponding attribute value derivation in the

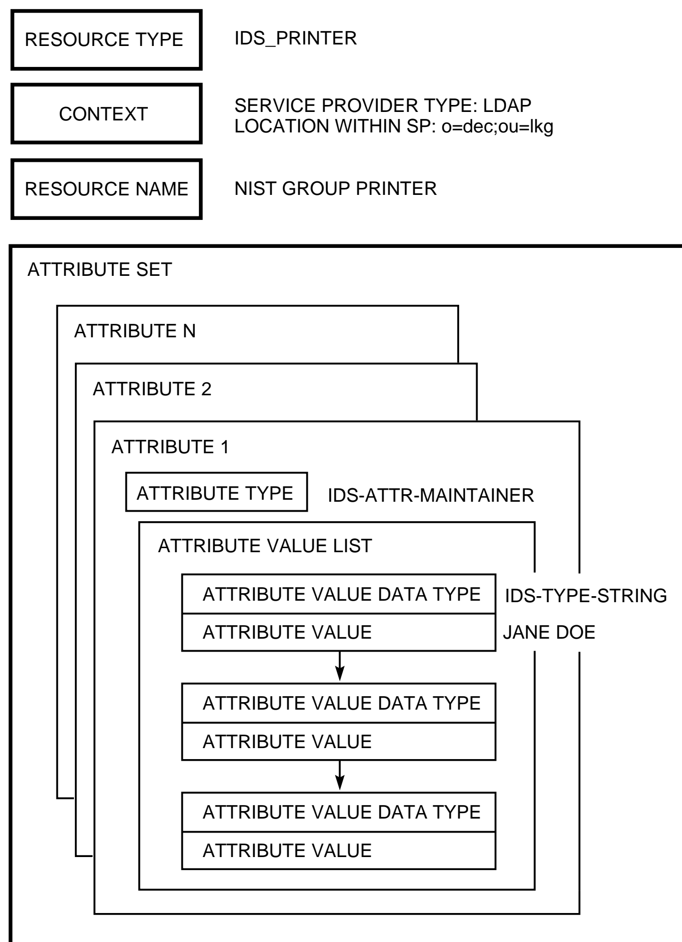


Figure 2
IDS Entry

IDS framework. This allows applications, and the IDS framework itself, to manipulate attribute values without knowing their types. The service providers, on the other hand, use the type information to translate from the IDS data formats to their native data formats.

Types To allow customers and third parties to identify their own IDS resources, the IDS type mechanism must uniquely identify objects. The two identifiers we considered using were universal unique identifiers (UUIDs) as defined by the Open Software Foundation Distributed Computing Environment (OSF DCE) and object identifiers (OIDs) as defined by the open systems interconnection (OSI) standards.[11,12] Some directory services identify attributes with OIDs, while others use UUIDs. For applications defining new resources, we wanted to avoid the necessity to obtain both an OID and a UUID. It is possible to encode a UUID in an OID, but the reverse is not true.

We could encode a UUID in an OID by registering an OID prefix. The prefix would indicate that the sequence after the prefix was a UUID. UUIDs are fixed-length structures generated from time stamps and Ethernet addresses, and therefore arbitrary information such as an OID cannot be encoded in them. UUIDs are also easier for application writers to generate because numerous systems ship with tools to generate them.

Certain directory services, for example X.500, have external type definitions for the directory entries. It is possible to define a generic entry and then map arbitrary values into that entry, but IDS entries would not be meaningful when viewed with the native directory management tools. We felt that this was unacceptable, because it would make the management of IDS entries in the namespace much more difficult. Some systems use UUIDs to represent the type information. We chose to use UUIDs since they are both easy to generate and can be used in both UUID and OID class definition systems. The use of OIDs would require UUIDs to be generated for UUID-based systems and mappings to be maintained.

Communities An IDS community is both an administrative grouping mechanism and a logical location for IDS resources. When people interact with the IDS system, they see a community as the organizing principle. The administrator controls the boundaries and membership of an IDS community. Typically, a community represents either a particular location such as a building or a functional grouping such as a work group.

Initially, we considered a supercontext to join multiple directories into a single logical directory. This supercontext would have contained multiple contexts, one for each type of resource supported by IDS. We eventually subsumed the supercontext into a community and called it a resource context list. An IDS community is stored as a special object in the directory. Each community's resource context list describes the directories that make up the community. The resource context list is the federation mechanism by which IDS determines where resources of each type are stored. Each entry in the resource context list is a pair of resource type and context. As users and applications operate on entries in a community, the IDS framework (through IDS entry and community methods) inspects the resource type and the community to determine the context. Figure 3 illustrates an IDS community.

One of the problems we anticipated was that large organizations would naturally tend to have many IDS communities: How would the user identify these? We considered an additional hierarchy in which communities would be members of other communities. Our usability consultants emphasized that users should not have to browse a hierarchy to access resources. In response, we developed the concepts of the local and the home community. The local community is associated with the machine a user is currently using—it represents a physical location. The home community is the one with which the user is associated or belongs. We envisioned that the home community would be the same as the local community at the user's normal place of work, but there is no requirement inherent in the design that things be organized this way. For example, if a user is associated with the community at her work site and the machine she uses is also located at that work site, both her local community and

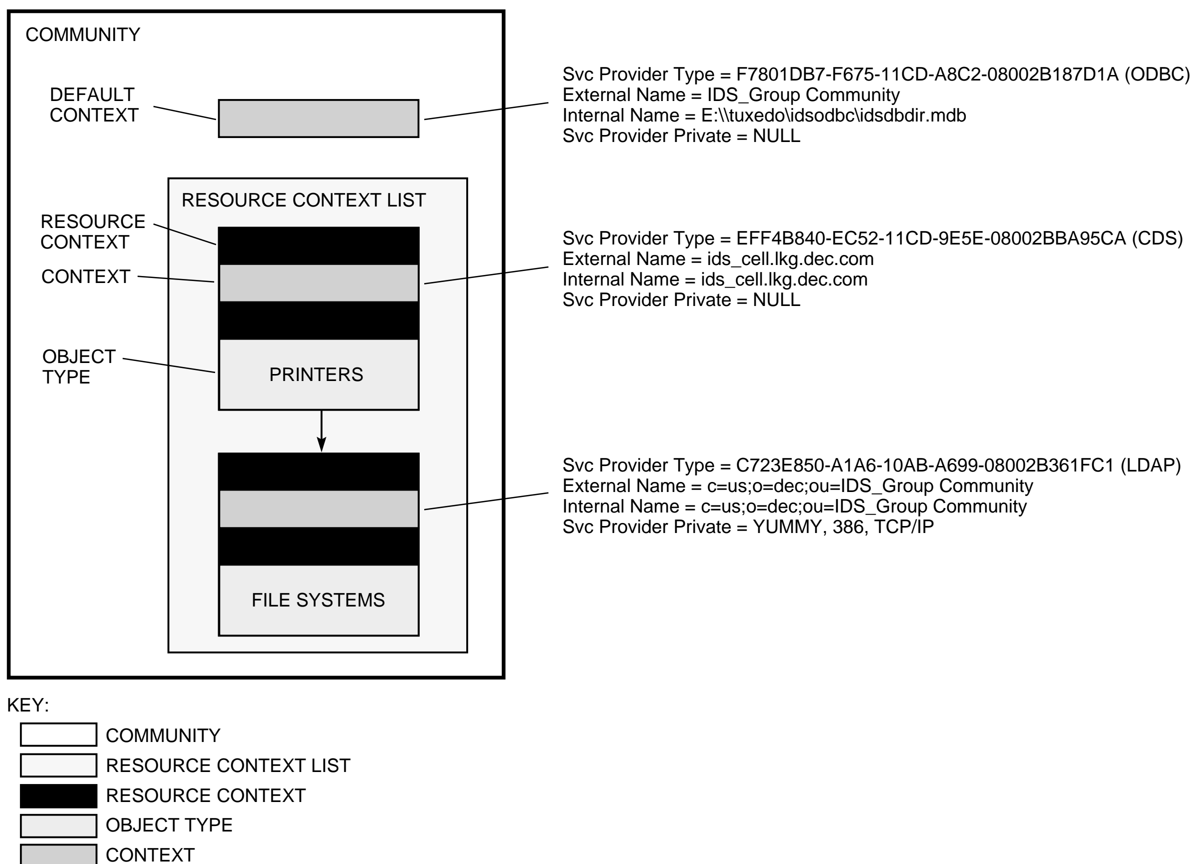


Figure 3
IDS Community

her home community represent this work site. If this user works at another work site and uses a different machine, her home community remains the same, but her local community reflects the community where the new machine resides. The concepts of local and home communities do not reduce the number of communities, but they do provide a direct method by which users can access the communities that contain the resources they most frequently use. The local and home communities are a convenience; users and applications are in no way restricted to those communities.

Search Support Searching is handled by the search object. The search object contains a community (or list of communities), a resource type, and an attribute filter. The attribute filter supports both equality and comparison matching of attribute values and allows callers to construct complex requests by concatenating comparisons together in a series of Boolean operations. For example, a caller could construct a filter that returned all printer objects that (((are located on Floor2) OR (are located on Floor3)) AND (support color printing)). Combined with the local and home community support, filters allow applications and users to express ideas such as "print this at the closest printer that supports color, two-sided printing, and then transmit it to any facsimile machine in my home community."

The search object's default filter returns all objects of the resource type in the local community. The search object resolves the community to a context and passes it to the service provider. The service provider constructs a list of matching IDS entry objects to return to the user. In IDS, the search object supports browsing.

The search object has methods that display a dialog and construct filters based on user input. When designing the system, we debated whether it was better for the search object to contain both the filter and the search dialogs or whether the filter construction belonged in the IDS entry. We chose to keep the search dialogs separate from the IDS entry. Experience with implementing resources derived from the IDS entry has shown this to be an error. Currently it is necessary to derive from two objects, IDS entry and the search object, to implement a resource that has a resource-specific search dialog. We will be modifying the search and IDS entry objects so that the construction of the filters and the dialog that constructs the filters are IDS entry methods.

Schema The service providers translate between the native directory object and the IDS entry. In general, directory service entries are not self-describing. In existing directory services, either a schema or the application is expected to know the directory-specific format of the data. The latter is more common than the former, and in any case the schema methodologies are unique to each directory service.

From the point of view of the native directory service, IDS is the application. To properly convert the data, the service providers must know what it is. The service providers use the schema to determine the correct attribute and value types to use when constructing the IDS entry of a particular type.

The schema describes resource types, attribute types, and attribute value data types. Logically, the schema is a set of tables, one for each service provider, which maps the native name or type to the IDS name or type. These tables are read by the IDS schema component when IDS is initialized. Because these tables are external to the system, they can be modified by users or applications.

There is one limitation on the extension of the schema: New attribute and resource types can be defined, but they must be composed from the predefined IDS attribute value types that the service providers can support. The service providers would have to be modified to support additional attribute value data types. This limitation is not as severe as it at first appears. A rich set of data types is defined in the existing directory services, and a relatively small set is in common usage. By defining the IDS data types to encompass the set of data types defined by existing directory services, we have reduced this limitation to a theoretical rather than a practical problem.

As a consequence of the use of schema, applications must specify the resource type for any IDS operation. This is a limitation that in principle does not exist in other directory systems. After some consideration, we concluded that few useful operations can be performed on an object whose type is unknown. To perform an operation on objects of all types, the schema can be interrogated for the list of all supported IDS object types, and the operation is then iterated over each type.

The System Object The system object loads and initializes the service providers. On initialization, the system object constructs a list of the available service providers from those defined in a local configuration file.

The system object constructs and maintains the list of known communities. The system object obtains this list using the following mechanisms:

- Inspect a well-known location (if one exists) to see if it contains a cache of known communities.
- For each service provider, call the discover method to ask the service provider for its list of known communities.
- If the system object is initializing for the first time, prompt the user to create a community.

Application Programming Interface

As mentioned previously, we divided the framework into an API and a service provider interface (SPI). The API consists of the search object methods, the IDS entry methods, the attribute object and value object methods, and the system object methods necessary to access communities.

Service Provider Interface

The SPI specifies the interface between the IDS framework and the native directory services. It defines the semantics for all operations that may be performed on IDS information regardless of which directory service stores the information. The SPI effectively insulates both the IDS framework and the IDS applications from the unique syntax and requirements of different directory services.

A directory-specific module, called a *service provider library,* provides a directory-service-specific implementation of all SPI operations and translates resource information back and forth between the IDS entry and the service-provider-specific format. A service provider library must be implemented for each directory service to be supported by IDS. Any directory service or information repository system that can provide the IDS SPI semantics may be an IDS service provider.

SPI Semantics The IDS SPI defines the following main operations: create, read, search, modify, discover, and delete. All SPI operations specify the name of the IDS community upon which to operate. Each IDS community maintains a list of contexts that specify in which service provider IDS resources of a particular type are stored and in what location within the service provider. The SPI uses this community name to retrieve the context information that directs the operation to the correct service provider library. With the exception of the delete operation, which requires an explicitly set context (to be sure that an explicitly located object is selected for deletion), if the caller does not set the community name, the local community is assumed.

The create, delete, modify, and read functions all operate on a single IDS resource at a time. Each, therefore, provides an IDS entry object to identify and/or describe the resource.

The create operation creates a new IDS resource in the directory. The create operation specifies the type of IDS resource to be created, the resource's name, and the IDS attributes and values associated with the resource. On a successful create operation, the service provider constructs a unique directory-specific name for the new IDS resource and stores this name in the object's IDS entry. The service provider subsequently may use this name to find the object more quickly rather than constructing it from the name, resource type, and context information contained in the IDS entry.

Before constructing the resource in the directory, the operation validates the IDS entry against the schema to ensure that it does not violate the schema. For example, attempting to create a resource without a required attribute value pair violates the schema and is flagged as an error. Conversely, the delete operation removes the IDS resource from the directory.

The modify operation updates the attribute and values associated with the resource in the directory. The modify operation supports the following update directives:

- Add a new attribute and value.
- Add a new value to an existing attribute.
- Replace a value of an existing attribute.
- Delete an attribute and its associated values.
- Delete a value from an existing attribute.

Each modify directive is verified against the schema before being applied to the directory.

A read operation retrieves a uniquely specified IDS resource from the directory, translates it into IDS entry format, and returns the IDS entry to the caller. The read function is typically used to compare the directory format of an IDS resource to one maintained in memory by an application, or to process IDS resources returned from a search operation one at a time.

The search function identifies and returns IDS resources that match the characteristics specified by the caller. To bound the scope of the search, the caller specifies the following search characteristics: resource type, community name or names to be searched, and a filter containing attributes and associated values or value ranges.

The discover operation is called by the IDS system object to find all communities known to a given service provider. Service providers for directory services that support a server solicitation and advertisement network protocol implement a discover function. In these directories, servers advertise their presence in response to network solicitation requests. The discover method uses the directory's native solicitation and advertisement protocol to discover local directory servers and then issues the appropriate operations to the server to determine if it has defined any IDS communities. Service providers that do not have a solicitation and advertisement protocol can implement an alternative discovery mechanism such as retrieving the community information from a file or provide no discovery mechanism.

Construction of the System: Directory, Session, and IDS Entry Objects The SPI is constructed of three framework objects: the directory object, the session object, and the directory operation methods of the IDS entry object. The directory object is responsible

for service provider initialization and termination, maintenance of session objects, and community discovery. Each service provider exports one directory object to the IDS framework. The session object implements all the directory operations on a service provider. Session objects are obtained from the service provider by means of the directory object. The IDS entry directory operation methods determine the context if it has not been set, obtain a session object from the proper directory, and dispatch the operation to the associated service provider through the session object. For efficiency, session objects may be cached by the service providers.

Implementation Considerations

Once we had established our basic approach, we turned our attention to implementation decisions.

Client versus Server

Our first consideration was whether to implement this technology as software executing on a server system or as software executing on a client system. The server solution had a number of attractive qualities: it would not be necessary to have all the native directory clients on all the desktops, and potentially complex processing would occur on an appropriate platform. However, we identified two problems with the server solution. The first concerned security. To access the directory service on behalf of a particular user, we would have to impersonate that client user on the server machine. Although this can be done without exposing security holes, doing so adds another layer of complexity to the problem. The second problem with the server solution was that it required the customer to find a machine for and deploy a server prior to getting started with the system. One of the design goals was to remove barriers to directory deployment, and we were concerned that a server solution would add a barrier. We saw a need for both client- and server-based solutions, and since the client solution was simpler to implement, we chose to start there.

Security

The IDS interfaces leave security to the underlying directory services; we did not attempt to abstract a general-purpose, access control or authentication interface. The primary reason for this was a conviction that the vast majority of current directory information is world read, and therefore a complex access control interface was not necessary. An access control and authentication layer that was directory-service-independent would have added significantly to the complexity of the project, and we chose to postpone this problem. IDS does pass requests directly to the native directory-service client; IDS does not alter or impersonate the user's identity. In that sense, it perfectly preserves the security inherent in the underlying directory services.

Filter Implementation

The implementation of the IDS attribute filter is based on the string filter as defined in RFC 1777.[13] The Lightweight Directory Access Protocol (LDAP) string filter provided a convenient internal representation, and we would be able to reuse the LDAP parsing and processing code that we had developed as part of an earlier product. We considered using SQL to construct IDS attribute filters, but chose not to do this for implementation convenience.

Service Provider Considerations

Initially, we thought that developing a directory-service-independent interface would not be difficult. Most of the required operations such as read and write are straightforward and obvious. The implementation of such an interface, however, proved to be difficult because the underlying directory services have, in some cases, very different native capabilities and semantics. We chose to implement service provider libraries for the following three types of service providers:

- Open Database Connect (ODBC)-compliant database
- X.500-based directory using the LDAP
- DCE CDS

These service providers are representative of the types of directories that exist today. Table 1 highlights some of the differences among the three directories. As this table illustrates, not all directories can natively support the semantics described by the IDS SPI. In these situations, we have followed three alternatives: (1) the service provider library implements the functionality, (2) the IDS framework implements the functionality, or (3) in a small number of cases, the service provider cannot implement the functionality and remains less functional.

Some operations cannot be supported natively by only one or a small handful of directory services. For these operations, we require the service provider developers to implement (or emulate as best they can) the functionality in the specific service provider library for that directory. For functions that a number of service providers cannot support or that are sufficiently difficult to implement, we provide a common implementation or emulation in the IDS framework that service provider libraries can call. For example, CDS does not natively support an attribute-based search mechanism. Rather than attempt to implement a CDS search capability, we chose to provide an IDS framework "prune" function that applies an IDS filter to a list of IDS entries and returns only those entries that satisfy all conditions of the filter. Service providers such as CDS can then

Table 1
Differences among the ODBC, X.500, and CDS Directories

Functionality	ODBC	X.500	CDS
Distributed directory service	No	Yes	Yes
Hierarchical organization of directory information	No	Yes	Yes
Attribute-based search	Yes	Yes	No
Attribute value-based search	Yes	Yes	No
Native schema support	Yes	Yes	No
User can extend IDS schema	No	Yes	No
Transactional semantics	Yes	No	No
Tolerant of intermittent connectivity	No	Yes	Yes
Provides security mechanism on connections	No	Yes	Yes

emulate the IDS search function by enumerating all resources of a particular type and then call the prune function to pare down the list of resources.

The IDS schema implementation is another example of a common capability we have provided for all service providers to use. Not all service providers support object, schema and, of those that do, fewer still can support user extension of the schema. We chose to allow user extensibility and implemented a service-provider-independent schema interface and mechanism.

In a few instances, we determined that it would be too expensive in terms of implementation time to provide a service-provider-specific or an IDS-framework implementation of an SPI-mandated function. In these cases, we allowed the service provider to remain noncompliant. For example, a call to initiate a session to a service provider specifies user name and password arguments. For those directories that support user name and password security mechanisms, we preserve that functionality. For directories such as the ODBC service provider that do not support these security mechanisms, however, we provide no additional security measures. The cost to implement and deploy such a security mechanism outweighs the gain of having the additional features.

In addition, we found that not all directories provide the same semantics for a particular operation. For example, when updating a resource, service providers handle existence checking of resource attributes differently. If requested to add an attribute value to an attribute that does not yet exist, one service provider returns an error, while another implicitly creates the attribute. We worked around problems of this type by carefully specifying the semantics and error conditions of all SPI operations. Service providers that do not natively support these SPI semantics must implement whatever additional functionality is required to do so. For example, the CDS service provider required additional functions that determined and flagged whether or not a particular attribute existed.

In addition to all errors that are specific to service providers, we return an error that is independent of any IDS framework service provider. This adds another level of consistency across our service-provider implementations.

Current Applications

As with any foundation technology, the proof of its viability lies with the applications that employ it. In the PATHWORKS product, we currently have three applications that use IDS:

- Network Connect
- IDS Administration
- Resource Synchronizer

The Network Connect application finds and connects users' printers and file shares. It provides a user interface that allows users to browse or search for file shares or printers. Through Network Connect, users can refer to resources by their logical name or their attributes. A single physical printer, with queues on several machines or several NOS systems, is presented to users as a single printer. Network Connect uses the IDS API to access the IDS search capabilities and to translate a printer or file share's IDS name to its network-specific name to connect to the resource. Network Connect may be accessed through the Windows version 3.1 Print Manager and File Manager utilities and through the PATHWORKS Network Connect utility.

The IDS Administration utility (IDS Admin) allows a network administrator to manage IDS resources and communities. IDS Admin is integrated into the Digital ManageWORKS Workgroup Administrator for Windows software product.[14] Admin creates, modifies, and deletes resources and communities. It also allows users to browse IDS resources and communities in the ManageWORKS hierarchy and to search for IDS resources.

An administrator can manage IDS resources manually through the ManageWORKS user interface or can rely on information provided through the semiautomatic resource collection utilities called the Resource Gatherer and Resource Synchronizer. The Resource Gatherer periodically collects information about network LAN Manager and NetWare printers and file shares. The Resource Synchronizer utility processes the gathered information, updating the directory. It also eliminates duplicate entries and discards information the administrator wishes to ignore. The gatherer and synchronizer allow the directory to be kept up-to-date, even if resources are added or removed through the native NOS interfaces.

Future Work

In the future, we plan to improve the IDS extensibility mechanisms. Currently, a local copy of the schema exists on every client. Propagating the changes to each client will become a problem as users and applications extend the schema. We are considering storing either the schema or a pointer to the schema in the directory.

The current IDS implementation runs on both the Windows version 3.1 and version 3.11 operating systems. We are currently porting it to Windows 95 and investigating ports to other operating systems, such as UNIX.

The implementation does not support the entire IDS design: Although resource context lists are implemented, there is no reasonable way for a user or administrator to create them. The user interface work for these features in the IDS Admin application has not yet been completed.

Summary

IDS provides a mechanism for integrating multiple directory services into a single system. It is predicated on the ability to define a common set of directory operations and on the type information. The implementation of three very different service providers—CDS, X.500, and ODBC—indicates that we succeeded in defining the directory operations. The use ofIDS in the PATHWORKS product shows that it does address the practical aspects of the problem of integrating multiple directories into a single, logical directory service.

Acknowledgments

We would like to thank the many past and present members of the IDS team who contributed to the design and implementation of the product. Special thanks to Konstantinos Baryiames, Anthony Hinxman, David Magid, Tracy Teng, and Tamar Wexler. We would also like to thank the members of the Directory Task Force, Dah Ming Chiu, Dennis Giokas, and William Nichols.

References

1. *CCITT Recommendation X.501* (1992) and *Information Technology—Open Systems Interconnection—The Directory: Models,* ISO/IEC 9594-2: 1992 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1992).

2. "Naming Concepts" in *Using NetWare Services for NLMs* (Provo, Utah: Novell, Inc., 1993).

3. *AES/Distributed Computing—Directory Services* (Cambridge, Mass.: Open Software Foundation, 1993).

4. "StreetTalk Naming Service" in *ENS Administrator's Planning Guide* (Westborough, Mass.: Banyan Systems, Inc., 1992).

5. "Microsoft Directory Services Strategy," a white paper from the Business Systems Technology Series (Redmond, Wash.: Microsoft Corporation, 1995).

6. *X/Open CAE Specification, Federated Naming: The XFN Specification* (Reading, U.K.: X/Open Company Ltd., 1995).

7. "Bindery Services" in *NetWare System Interface: Technical Overview* (Provo, Utah: Novell, Inc., 1990).

8. S. Radicati, "Implementing the DIT" in *X.500 Directory Services: Technology and Deployment* (New York: Van Nostrand Reinhold, 1994).

9. P. Mockapetris, "Domain Names—Concepts and Facilities," Internet Engineering Task Force, RFC 1034 (November 1987).

10. P. Mockapetris, "Domain Names—Implementation and Specification," Internet Engineering Task Force, RFC 1035 (November 1987).

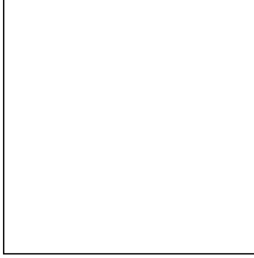
11. *AES/Distributed Computing—Remote Procedure Call, Appendix A* (Cambridge, Mass.: Open Software Foundation, 1993).

12. *CCITT Recommendation 208* (1992) and *Information Technology—Open Systems Interconnection—Abstract Syntax Notation One (ASN.1)* ISO/IEC 8824-2:1992 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1992).

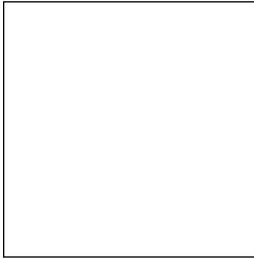
13. W. Yeong, T. Howes, and S. Hardcastle-Kille, "X.500 Lightweight Directory Access Protocol," Internet Engineering Task Force, RFC 1777 (March 1995).

14. D. Giokas and J. Rokicki, "The Design of ManageWORKS: A User Interface Framework," *Digital Technical Journal,* vol. 6, no. 4 (Fall 1994): 63–74.

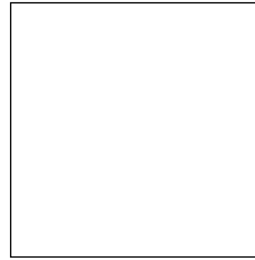
Biographies



Margaret Olson
Margaret Olson is a consulting software engineer in the Network Software Group. She was the project and technical leader for the IDS development project. For the last six years, she has had technical leadership roles in Digital's Directory Services Group. Before joining Digital in 1989, she worked in the networking and distributed computing areas at Apollo Computer. She received a B.A. (Sigma Xi) from Wellesley College in 1981. She published a paper on network licensing in 1988.



Laura E. Holly
Laura Holly is a principal engineer with the Network Software Group. She was a key technical contributor to the IDS development effort. Laura has previously contributed to the areas of DCE, distributed system, and knowledge-based system development. She joined Digital in 1985 after receiving an A.B. (high honors) from Smith College. Laura holds a patent and has published several papers in the area of knowledge-based systems.



Colin Strutt
Colin Strutt is a consulting software engineer and technical director for Teaming Software in the Network Software Group, where he is helping to define new PC-based software products. Previously, he has held technical leadership roles in directories, network management, and terminal server development, and before that led product developments in Ethernet servers and DECnet. He joined Digital in 1980 from British Airways in the U.K. He received a B.A. (honours) in 1972 and a Ph.D. in 1978, both in computer science from the University of Essex, U.K. He is a member of BCS and ACM. He has two patents issued and several patents pending and has published extensively, particularly on management technology.