

Integration of VIGO into Directory Services

Martin Wollschlaeger

Abstract

Modern networked systems are organised using directory services. These directories implement hierarchically organised object oriented components describing capabilities and functionality of the different networked devices. In addition, interfaces for accessing the directory objects are defined. The paper shows different implementation concepts for directory services using COM- and JAVA-based technologies. A concept for the integration of VIGO into directory structures is described. The mapping of the VIGO components to directory objects and the integration of VIGO interfaces into the interface definitions of directory services are shown. Aspects of directory-enabled applications are discussed. Furthermore, the relation of directory-supported VIGO systems to intranet-based management solutions are discussed. Future trends are shown integrating fieldbusses into heterogeneous communication networks using directory services, problems and benefits for users and vendors are discussed.

1. Introduction

Modern computer networks offer a large number of functions. These functions are distributed across the network. The single networked components are the provider for the functions. A consumer that wants to use the functions in a network needs to know, which specific functions are implemented in the network. Furthermore, the consumer needs to know where these functions are implemented, and which parameters have to be used for calling the functions.

During the years, different scenarios have been developed to overcome the problem described above. They range from centralised solutions with a single database to totally decentralised solutions, where all the components implement their own device specific database. One of the most interesting solutions is a directory containing a description of all the services and functions implemented in the components across the network. This directory needs to have an unique interface for requesting and searching for data sets, that can be easily accessed from any point within the network. As the name "directory" implies, it is organised similar to the well-known directory structure on a computer's hard disk, but it has a lot more capabilities.

2. Directory Services

Typically, a directory consists of a number of objects. These objects are mapped to the functions and the resources they have to describe. The whole directory is organised as a tree (**Fig. 1**). It consists of container objects and leaf objects. While leaf objects contain the descriptions, container objects act as containers for the leaf objects or for other containers. Typically, the containers are used to build up a structured system. There are different strategies for splitting the tree, e.g. geographical structures or function oriented. By structuring, each leaf object is assigned a

Dr.-Ing. Martin Wollschlaeger
Institut für Prozeßmeßtechnik und Elektronik (IPE)
Otto-von-Guericke-University Magdeburg
P.O. Box 4120
39016 Magdeburg
GERMANY

Tel.: +49 (391) 67-1 46 53
Fax: +49 (391) 5 61 63 58
e-mail: mw@ipe.et.uni-magdeburg.de

specific context. Some of the properties are valid only within this context, others are valid within the whole tree.

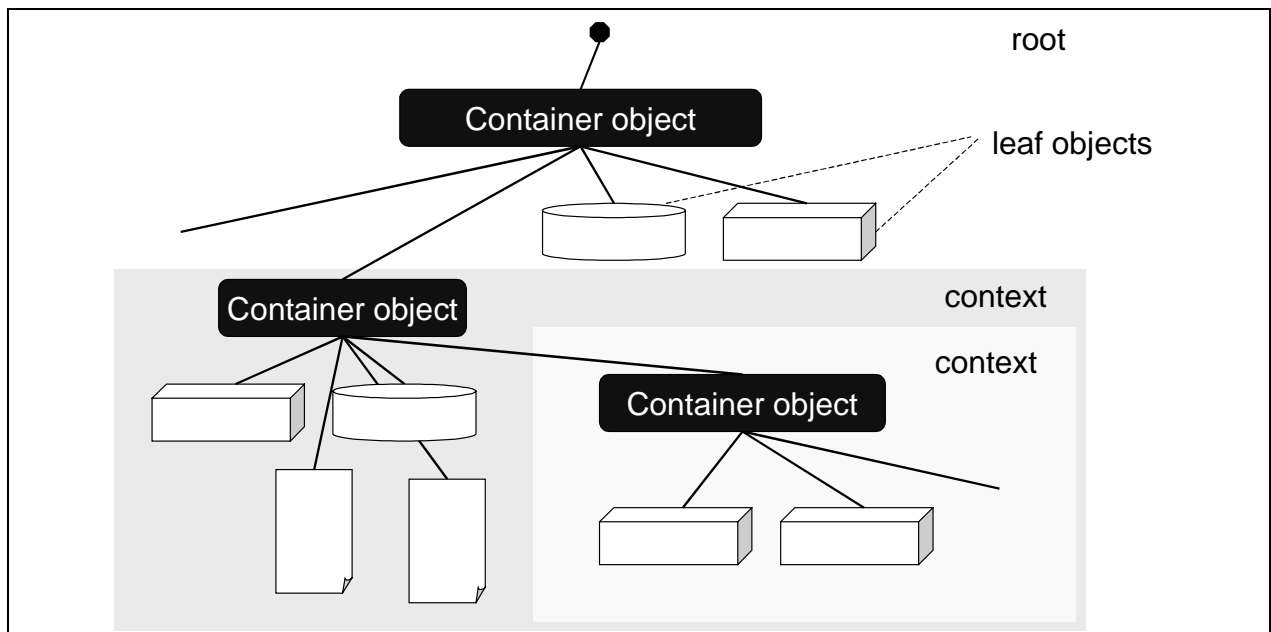


Fig. 1: Principle structure of a directory tree

Fig. 2 shows the central role, that a directory plays within a system. Different services use the information stored in the directory. The queries are performed using a directory access protocol (DAP). This protocol specifies the attributes of the objects in the directory and defines the functions used to parse the directory /1/.

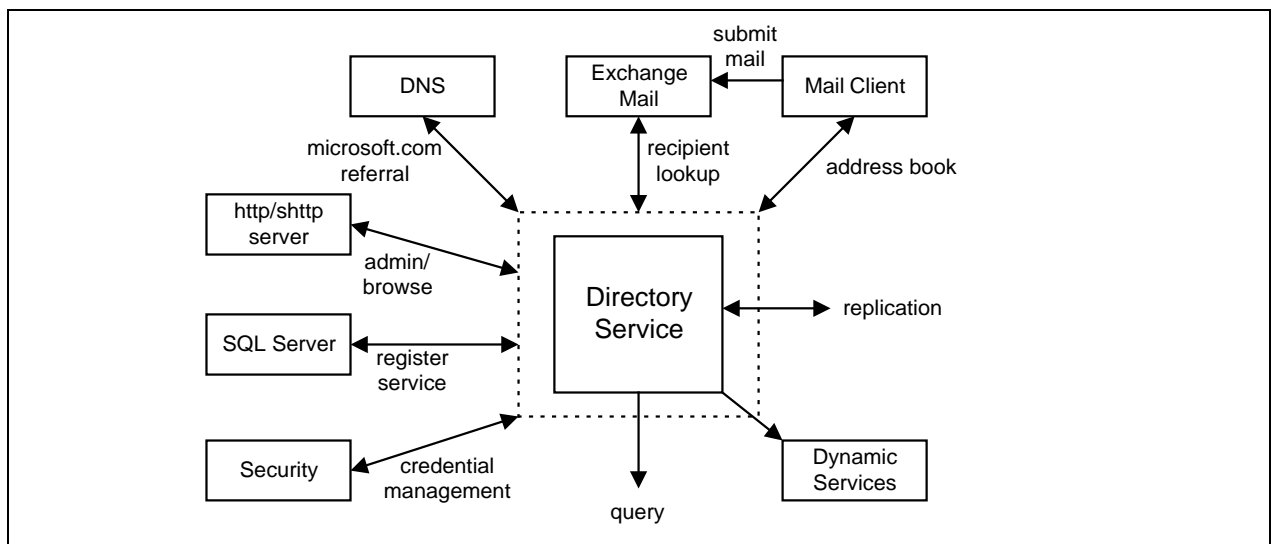


Fig. 2: A directory service and clients

Since different directory structures have been developed during the years, there are inconsistencies in the functions for directory access. This is dedicated to the different information providers implementing directory services on different operating systems. An interesting opportunity to overcome this problem is an integration of the single directory service providers within a single environment using a single implementation concept. Unfortunately, there are once again two different solutions – Microsoft’s Active Directory (**Fig. 3**) /2/ and the JAVA-based JNDI (Java Naming and Directory Interface) /3/. While Active Directory will be implemented in the Windows 2000 family, JNDI implementations are available right now.

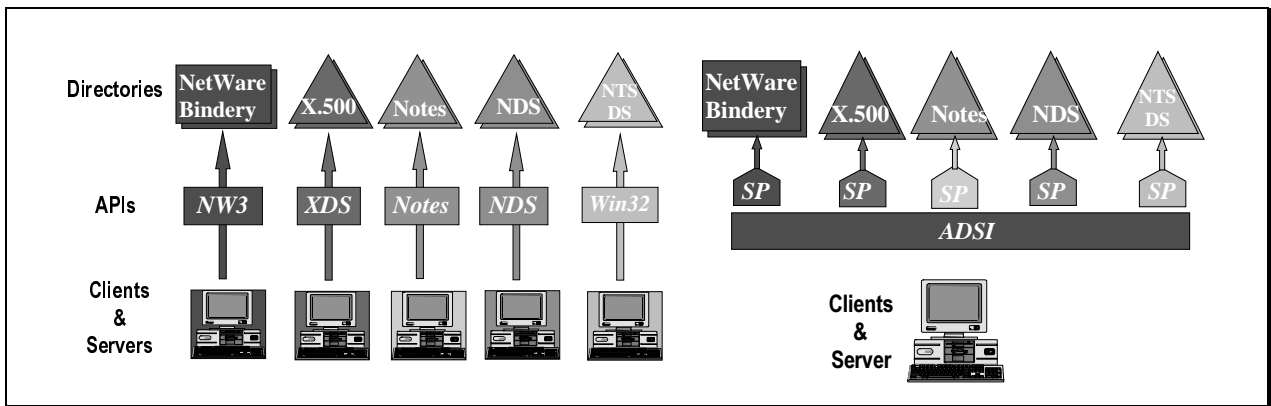


Fig. 3: Active Directory Services

Using Microsoft's approach, the directory access is performed by means of the Active Directory Service Interface (ADSI), an integration layer that harmonises the different information sources. Typically, a user does not explicitly recognise the existence of a directory service. The various information providers are covered by a tool, that uses the directory to gather information on the underlying services (**Fig. 4**). Such a tool presents a common user interface for manipulating the resources beneath.

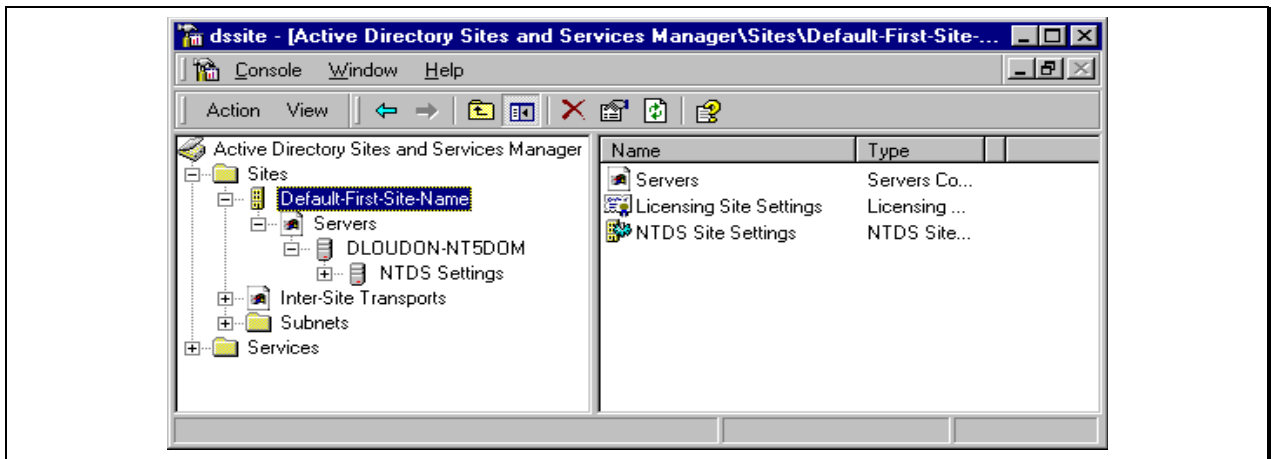


Fig. 4: Example of a directory-enabled tool

ADSI is implemented using COM technology [4], [5], providing easy access to the objects for COM-supporting programming environment. **Fig. 5** shows the principle structure of clients and servers using ADSI.

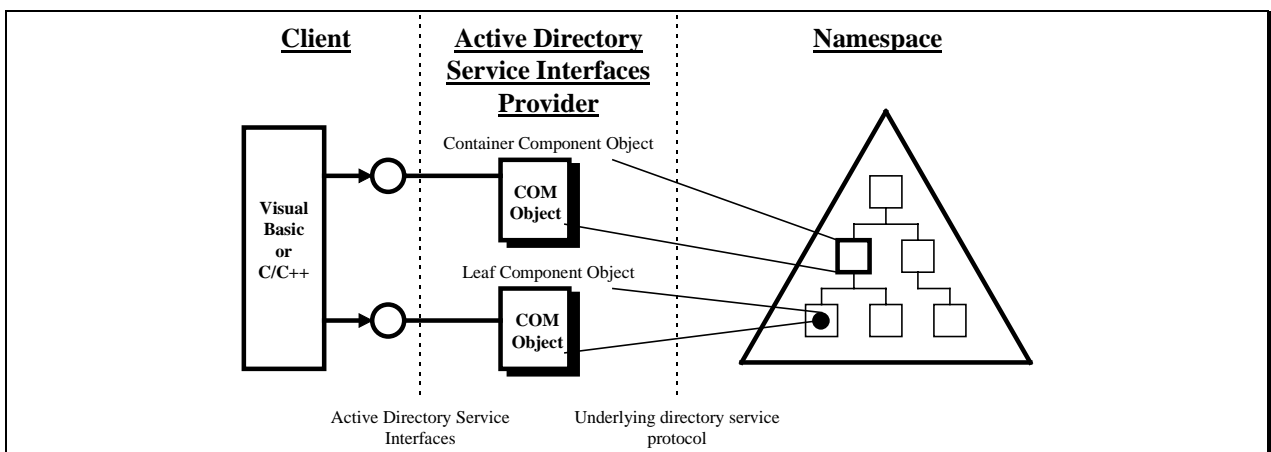


Fig. 5: Principle structure of ADSI client and servers

ADSI supplies a set of predefined objects and interfaces /6/ so that client applications can interact with directory services using a uniform set of methods. However, ADSI may not provide access to the complete functionality of a particular directory service. To help use the full power of each directory service, ADSI supplies a schema model that directory service providers and third-party software vendors can use to extend functionality beyond the interfaces ADSI offers.

3. Fieldbusses and Directory Services

One of the most interesting recent developments in fieldbus technology is the trend towards an integration of fieldbus systems into LAN structures. This enables the use of network-oriented tools and technologies in the fieldbus area. So the fieldbus has become an integral part of an industrial communication network, providing special features like real-time capabilities and specialised functions implemented in specialised components.

Searching for another statement characterising the trend described above, it can be stated, that a fieldbus has become a special information provider within a global (industrial) network. In order to manage the fieldbus components as integral parts of all the networked components of a system, their management functions have to be integrated into network-wide usable management tool. Therefore it is necessary to provide information on the components for the tools.

Using directory services, the components are accessible for all management tools using directory based information. Requests can be made to search the whole directory for the components, to gather information on the components and to find and install dedicated functions into application frameworks.

The integration of fieldbusses into directory services can be performed by creating a new namespace. A namespace describes basic features and attributes that are unique to all objects within a specific context. The fieldbus information provider extends the information providers shown in **Fig. 3** by introducing generic fieldbus components (**Fig. 6**). These components are described by specific attributes common to all fieldbus networks. Examples for such attributes are the assigned logical name of the component, its address, its network name and number, manufacturer information and so on (**Fig. 7**).

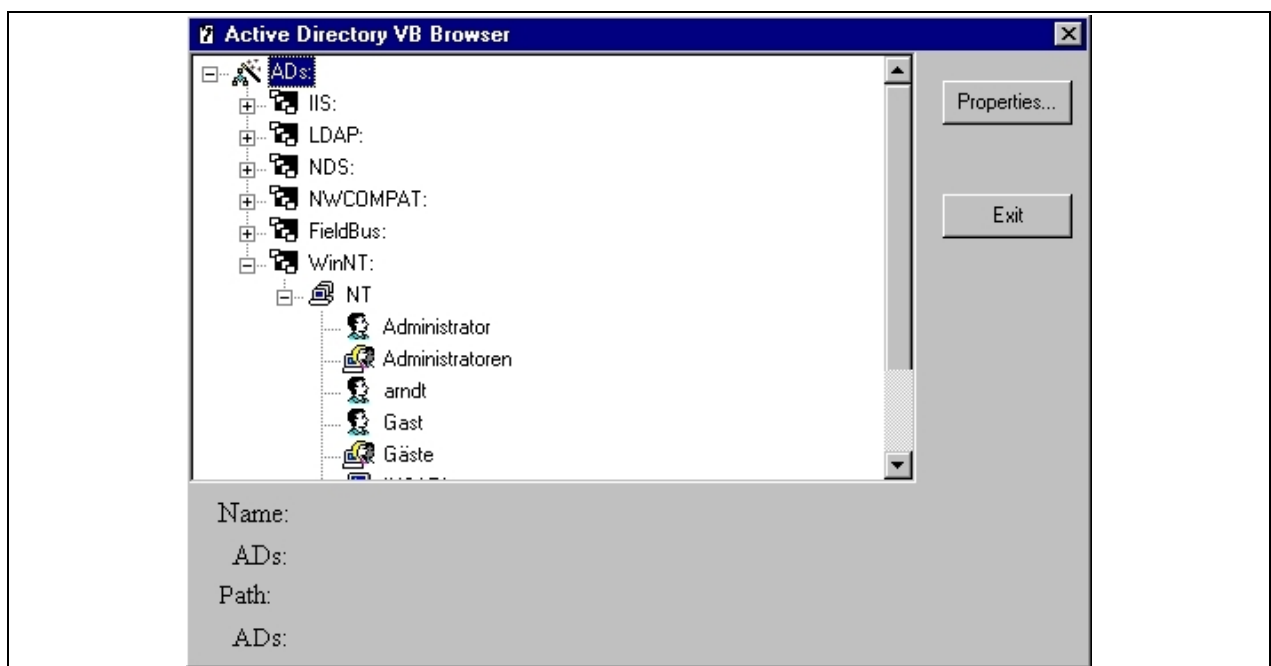


Fig. 6: Extending ADSI information providers with fieldbus information

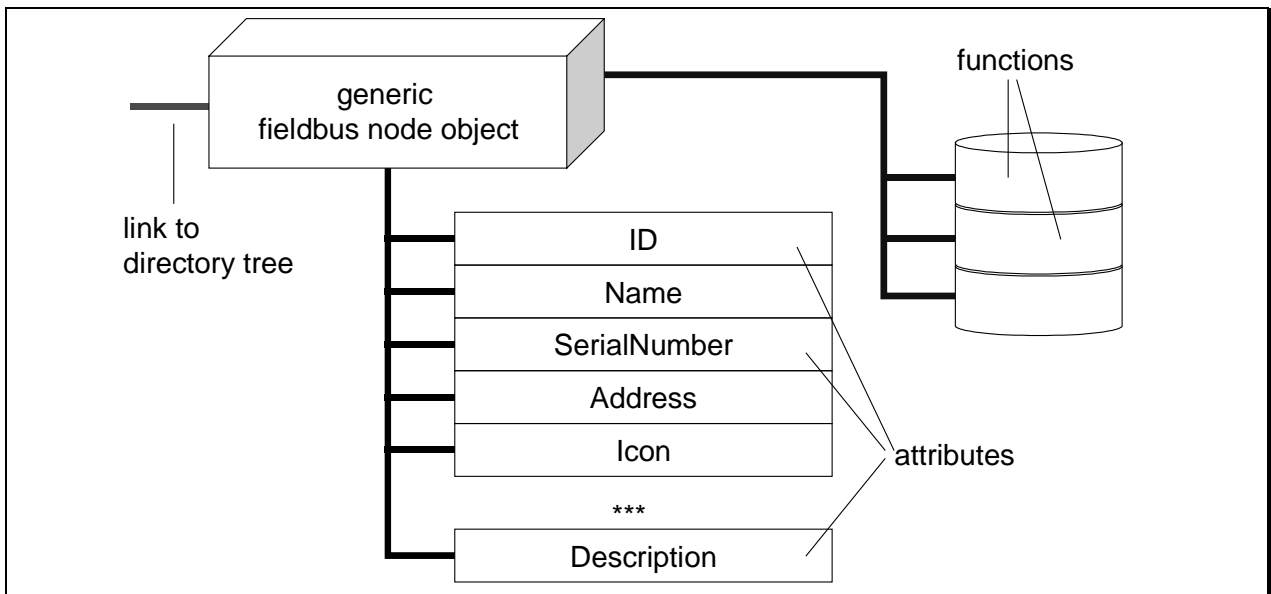


Fig. 7: Generic fieldbus object

A special feature of fieldbus component is the existence of specialised management functions that can be integrated into application or management frameworks. An example for such a management function is the channel configuration utility defined in VIGO. These management functions can be described as a “special feature” of the directory object. These functions then can be searched for, can be uploaded and installed and are ready for use in the framework.

There are different concepts for structuring the directory tree. One possibility is a topological structure describing the projects, networks, devices, device functions and so on. Another chance would be a functional oriented description, using layered automation functions independent of their real implementations. Both methods are suitable to describe an automation project, while the first one is more traditionally oriented, the last one covers the modern distributed systems’ approach (**Fig. 8**).

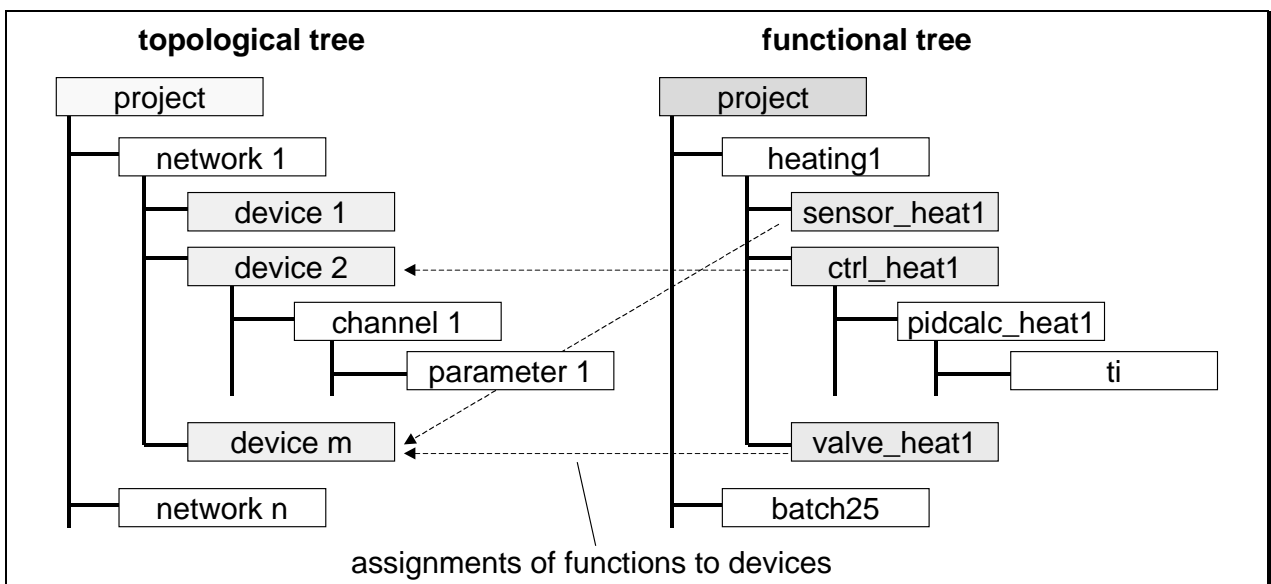


Fig. 8: Possible methods for logical directory structuring

4. Integration of VIGO

4.1. Integration concept

P-NET is one of the few fieldbus systems providing a unique, hierarchically organised information description method for the whole system. This description is defined in the VIGO database [7]. In order to make the description accessible from a directory service, the VIGO database has to be integrated into directory service objects. Preventing the efforts and problems concerning double-hosted information stores, the integration has to leave the physical implementation of the VIGO database unattended. This means, that a redirection of the directory service information requests to the VIGO database has to be performed. This redirection uses the functions implemented in VIGO for gathering the desired information (**Fig. 9**).

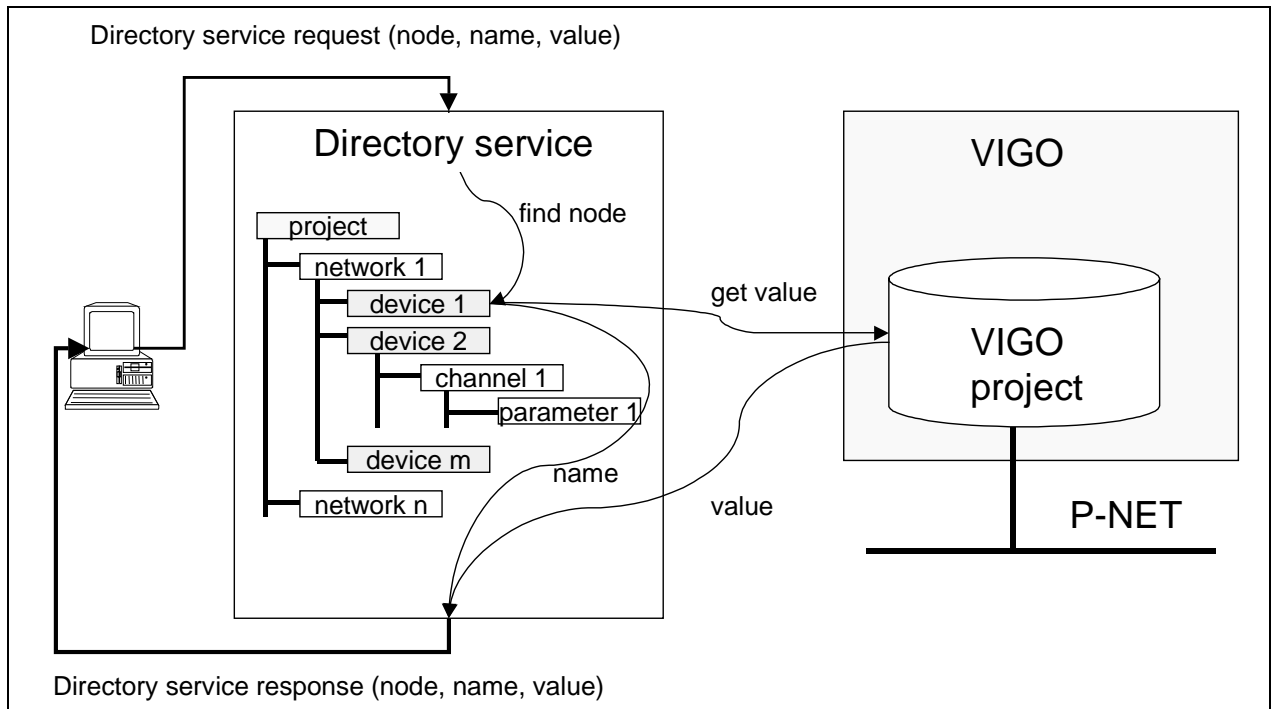


Fig. 9: Redirection of a directory service request to the VIGO database

4.2. Mappings

The best suitable solution for the mapping is a one-to-one-relation between the VIGO object classes (e.g. network, device, channel) and the appropriate objects of the directory tree. Therefore new directory objects have to be defined. These objects are implemented using the fieldbus namespace and provide specific information describing the special features of the P-NET components.

These directory objects have to be introduced in order to map the directory access protocol to objects redirecting queries to VIGO. Therefore these objects act as clients to VIGO, using the VIGO interfaces to access the VIGO database (**Fig. 10**). However, these mappings are only used, if the directory object does not contain the desired information itself. For example, information additional to VIGO are not included into the VIGO database, but are contained by the directory object itself. Therefore a redirection of the request is not necessary. The directory object will provide this information itself.

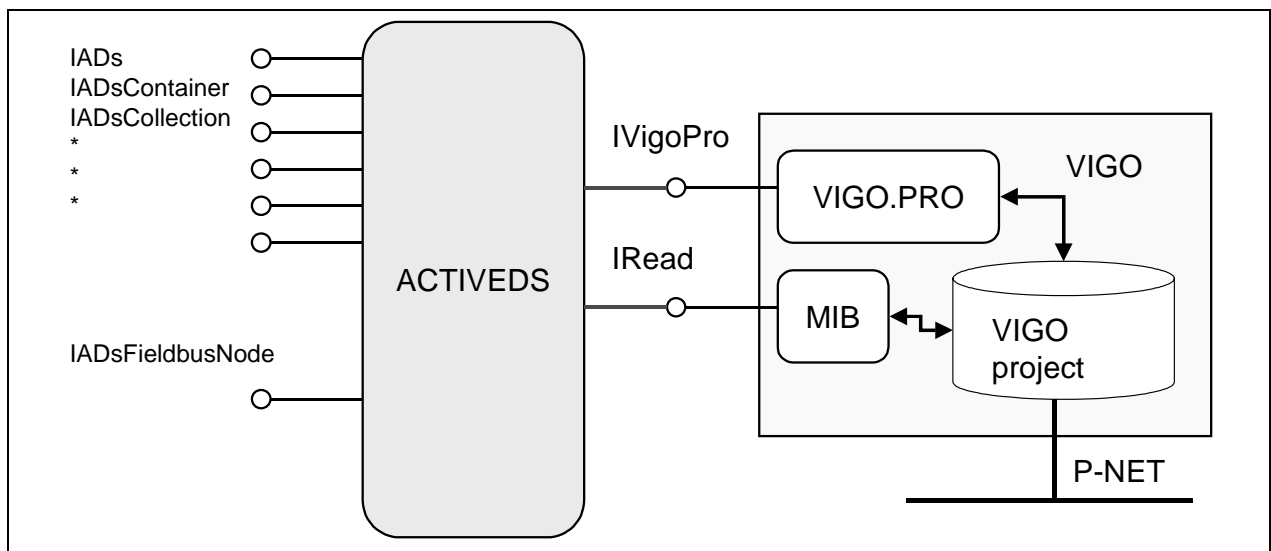


Fig. 10: Mapping of VIGO objects to directory service objects

In addition, all the information concerning the management of projects is covered by a special directory service object class. This guarantees, that project manipulation tasks are integrated into directory services as well.

4.3. Interfaces

As described above, the objects have to implement different interfaces. They have to provide the information requested by the directory access protocol. Using ADSI, they have to implement an interface derived from the ADSI definition. These interfaces are dual interfaces, allowing easy access from dedicated clients written in C++ as well as from clients using Dispatch interfaces (OLE Automation clients). The opportunity of “speaking both languages” enables any OLE-Automation-capable environment, including scripts, to access the directory objects.

On the other hand, the objects have to act as clients to VIGO. That’s why they use the functions provided by VIGO’s object definitions. The directory objects mapping VIGO objects build up instances of the vigo.pro object and make these instances pointing to the appropriate VIGO database entries by assigning the PhysID. Then it uses the Ivigo.pro functions to access the information stored in VIGO. The list of PhysIDs is generated during a start-up procedure, that retrieves the PhysIDs from the VIGO project. During this start-up procedure an instance of the MIB object is created, pointing to the VIGO project. The functions of the IRead and ISuperior interfaces are used to parse the project and to generate the reference list between directory objects and VIGO objects.

5. Application Aspects

Active Directory Service Interfaces abstract the capabilities of directory services from different network providers to present a single set of directory service interfaces for managing network resources. This greatly simplifies the development of distributed applications, as well as the administration of distributed systems. Developers and administrators use this single set of directory service interfaces to enumerate and manage the resources in a directory service, no matter which network environment contains the resource. Thus, ADSI makes it easier to perform common administrative tasks, such as adding new users, managing printers, and locating resources throughout the distributed computing environment, and ADSI makes it easy for developers to “directory enable” their applications /8/.

ADSI is designed to meet the needs of traditional C and C++ programmers, system administrators, and sophisticated users. With ADSI, development of directory enabled applications is fast and easy. ADSI presents the directory as a set of COM objects, which provide behaviour in addition to data.

5.1. Directory-enabled Applications

Users can get the most benefit, if they use directory enabled Applications. Common examples for this class of applications can be found in administration tools of networks, for example Netware Administrator. These directory-enabled applications use the C/C++ interface of the directory service to navigate through the objects and open the assigned properties for manipulation.

Using this concept, any directory-enabled tool can be used for information and management purposes covering the whole networked system. This provides the ability to the user to stay within his well-known user interface and management framework while performing management tasks.

Fig. 11 shows an example using a VIGO project within a directory tree of a Netware Network /9/.

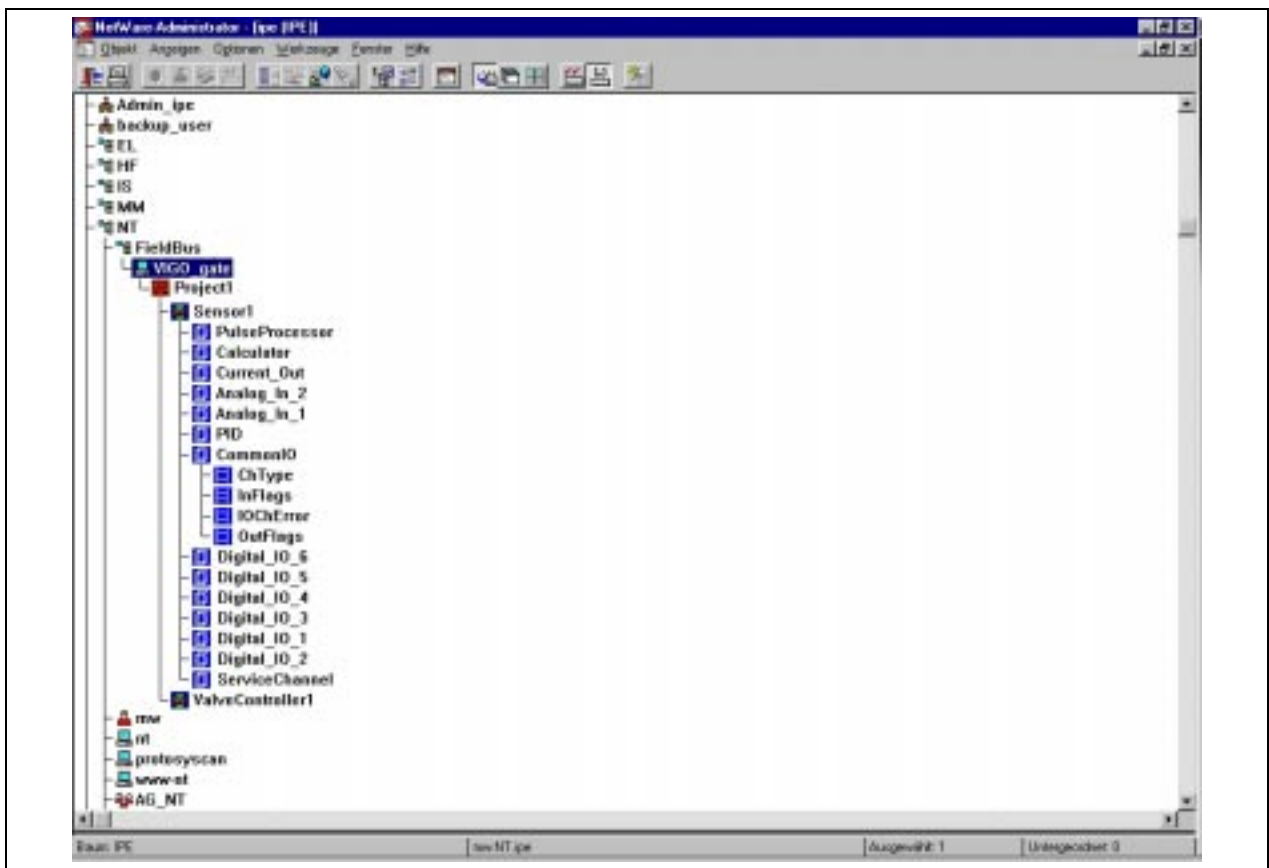


Fig. 11: Example of a VIGO project within a directory-enabled tool

5.2. Intranet-based Management

Another outstanding trend in management of networks is using Intranet-based concepts. The directory as a central information source can be accessed via scripts /10/. These Scripts use the IDispatch interface of the ADSI implementation. Besides scripts, any automation-capable programming language like Visual Basic or Delphi can use this interface.

Building lists of nodes in a project and their properties is a common need. In the example shown in **Fig. 12**, a Visual Basic script extracts all nodes in the “VIGO” namespace in the "Dir_test" installation. Here each node’s name and serial numbers (as they appear in the directory) are passed to a “PrintNode” routine.

```
dim MyNodeContainer as IADsContainer
dim MyNode as IADsFieldbusNode

set MyNodeContainer as GetObject("VIGO://Dir_test")

for each MyNode in MyNodeContainer
    PrintNode MyNode.Name, MyNode.SerialNumber
next MyNode
```

Fig. 12: Code fragment of a script printing node information

By implementing these scripts on a server, they can be used to manage the components within a given network. Based on directory information, common tasks can be performed without having to change the scripts depending on the context, and without having to provide specialised scripts for different component types. So a script enumerating the computers of a given network and assigning them to a graphical map can be used for doing the same things with printers or even fieldbus components without changing or modification. This will reduce the efforts for creating software for management frameworks.

6. Conclusion and trends

Further developments can be expected addressing the problem of managing distributed, heterogeneous networked installations. Directory services will play an important role as independent information providers across the network. Especially for the developers of management solutions and software, a single method for accessing distributed information is a key feature for reducing efforts and costs in software design and programming. The same statement can be related to the developers of fieldbus systems without any restriction. Implementing directory services enables the developer or manufacturer to provide an easy to use management solution, that can be integrated in whatever (directory based) system a user prefers to use. For the user, this flexibility in choosing a suitable framework is the key feature. There is no need to get familiar with a complex, special-purpose software system. The user interface is the same as in network management software used before. Of course, there have to be special properties, but this is not in conflict with the general benefit described above.

In addition to the effects discussed, directory services provide a good scalability and security. Further trends, like application frameworks or general-purpose description techniques of networked components, are expected to rely on top of information providers. Directory services will contribute to those developments as well.

References

- /1/ n.n.:
Microsoft Windows NT Active Directory: An Introduction to the Next Generation Directory Services.
White paper, Microsoft, 1999.
- /2/ n.n.:
Active Directory Technical Summary.
White paper, Microsoft, 1999.
- /3/ <http://java.sun.com/products/jndi/index.html>
- /4/ Brockschmidt, K.:
Inside OLE.
Second Edition.
Microsoft Press, 1997.
- /5/ Chappel, D.:
Understanding ActiveX and OLE.
Microsoft Press, 1996.
- /6/ n.n.:
Microsoft® Active Directory Service Interfaces: ADSI.
White paper, Microsoft, 1999.
- /7/ Cramer, O.:
VIGO, a Fieldbus Management System.
4th International conference on the P-NET Fieldbus system, Oporto, May 2nd - 3rd 1996, proceedings.
- /8/ n.n.:
Active Directory Programmer's Guide.
White paper, Microsoft, 1999.
- /9/ Andrew, C. et al:
NDS Developer's Guide.
Novell Press, San Jose, 1999.
- /10/ Hahn, S.:
ADSI ASP Programmer's Reference.
Wrox Press Ltd, Birmingham UK, 1998.