WOSA

Windows Open Services Architecture

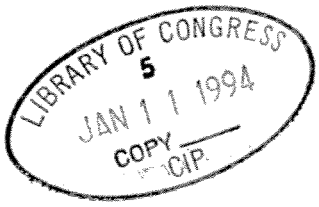
Computer Technology Research Corp.

WOSA

Windows Open Services Architecture

Jerry Cashin

Computer Technology Research Corp.
6 North Atlantic Wharf, Charleston, South Carolina 29401-2150 U.S.A.
Telephone: (803) 853-6460 • Fax: (803) 853-7210 • Telex: 147195



WOSA: Windows Open Services Architecture

First Edition – 1994

ISBN 1-56607-018-X

Published by Computer Technology Research Corp., Charleston, South Carolina U.S.A.

Editor – *Brian J. Lindgren*
Assistant Editors – *Laura H. Howe, Nancy Wagner*
Editor Assistant – *Roberta Duck*
Graphic Artist – *Nancy Wagner*

Library of Congress Cataloging-in-Publication Data

Cashin, Jerry.
WOSA : Windows Open Services Architecture / Jerry Cashin -- 1st ed.
p. cm.
ISBN 1-56607-018-X : $190.00
1. Windows (Computer programs) 2. Computer architecture.
I. Title.
QA76.76.W56C377 1993
005.4'3--dc20

93-34096
CIP

WOSA
Windows Open Services Architecture

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

LIST OF TABLES

Introduction

Windows Open Services Architecture (WOSA) is a natural evolution in the quest to help users integrate information from a wide variety of sources and platforms. PC users are no longer content solely with standalone capabilities. They must also have access to all major information resources in an enterprisewide computing environment. With increasing specialization, however, this task has become more difficult. WOSA provides one approach to the problem of universal access. There are alternative solutions, such as those from the Open Software Foundation (OSF), Unix International (UI), Apple Computer, and others.

The Distributed Computing Environment (DCE) from OSF, for example, provides a middleware solution. Atlas from UI, and Open Network Environment from Apple represent different techniques for integrating enterprise services. DCE's interface conventions are not directly tied to the operating system. Rather, there is another layer of control that functions, for the most part, independent of the resident operating system.

WOSA, of course, works closely with Windows. This is both a plus and a minus in terms of producing effective results. On the one hand, efficiency is enhanced by close linkage with the operating system. On the other hand, questions of application portability outside the Windows environment are a legitimate concern.

The great attraction of WOSA to Windows software developers is that standardization of the interface to multiple software services enables their product to reach a wider audience. If, for example, a front-end database access product follows WOSA interface conventions, it will be

able to interact with various database offerings, as long as the latter also follow WOSA dictates.

WOSA employs a Windows Dynamic Link Library (DLL) that permits software to be linked at runtime. This allows applications to dynamically connect to services. Applications call protocols known as Application Programming Interfaces (APIs) to access services that have been standardized in the Windows environment. The specific nature, configuration, etc. of the called service is of no concern to the calling API, at least from the viewpoint of access procedures.

In Microsoft's world, WOSA represents an important milestone. Even in competing constituencies, some aspects of WOSA may gain de facto approval. Two early candidates for the latter are Microsoft's messaging and database interfaces. This whole arena is a fast-changing landscape, however, and ultimate acceptance of WOSA APIs outside the Microsoft firmament is uncertain. There is no doubt that acceptance within the Windows community will be almost total.

The focus of this report is to identify and analyze the major components of WOSA. It also reviews ancillary issues that impact developments in this area. The report is divided into 11 chapters:

1 – Executive Summary
2 – Windows Operating System
3 – WOSA Architecture
4 – Open Database Connectivity
5 – Messaging API
6 – License Service API
7 – Windows Sockets API
8 – Windows SNA API
9 – Windows Extensions for Financial Services
10 – Windows Remote Procedure Call API
11 – Windows Environment Trade-Offs

The Executive Summary provides an overall assessment of the major components and issues associated with WOSA technology. The various

hardware and software options are reviewed, along with the associated support tools.

Scratch the WOSA surface and there will be a Windows platform in the equation. Whether 3.x or NT, Windows is the "bedrock" of WOSA technology. Chapter 2 reviews the Windows operating system and its relationship to the WOSA phenomenon.

Windows' widespread use guarantees that any architecture based on its tenets will occupy an important place in mainstream computing. WOSA has received much attention due to its Windows association. Chapter 3 reports on the architecture of WOSA, and compares it to the OSF's DCE.

Any solution devised to support enterprise computing must deal with the multitude of databases and formats in the marketplace. Chapter 4 looks at Open Database Connectivity (ODBC), which is designed to facilitate access to various database products from the same application.

Any major architectural innovation must have a messaging component as part of its technology mix. WOSA is no different in this regard, offering Messaging API (MAPI) for support of the electronic mail (E-mail) function. Chapter 5 evaluates MAPI and some competing messaging systems.

Chapter 6 analyzes the License Service API (LSAPI) feature in WOSA. Software licensing has become a burdensome management problem to many processing sites. LSAPI alleviates some of this burden by providing a standard interface to diverse licensing utilities.

Windows Sockets API offers a gateway to Unix technology from Windows platforms. It is an important marriage of client to server. The basics of Sockets, along with extensions appended for the Windows environment, are explained in Chapter 7.

Another important Windows linkage is to IBM's ubiquitous Systems Network Architecture (SNA). Microsoft's SNA API operates from an

SNA Server which provides a standard interface to this vast networking resource. Chapter 8 details the nature of this interface.

The world of banking and related financial services presents unique problems related to special hardware and software requirements. Chapter 9 explains Windows Extensions for Financial Services, including various implementation issues.

Building distributed applications often involves the use of Remote Procedure Calls. Chapter 10 reviews the specifics of this important networking capability.

With any relatively new architecture such as WOSA, there will be trade-offs, product comparisons, and apparent limitations. Chapter 11 looks at many of these elements in the WOSA environment.

Chapter 1

Executive Summary

WOSA Background and Overview

The search for Windows Open Services Architecture, or something with the same general capabilities, began when the first personal computer was connected to a mainframe. Since then, information systems (IS) managers have been working to improve the synergy among diverse computing resources and associated data storage modules.

Most organizations are striving to achieve a high level of collaboration from their computing and information resources. As technology has advanced, however, and specialization has increased, the task has become more difficult.

Many sites have installed PC local area networks (LANs) in order to attain greater workgroup cooperation. A preponderance of these configurations provides file and print services. This is a step along the path to sophisticated groupware participation and interaction, but offers nothing like the full range of sharing and interoperability sought by IS managers.

WOSA's ultimate goal is to allow Windows-based applications to enjoy seamless access to all available information without having to know anything about the underlying infrastructure, i.e., the type of network, computer, or back-end services. Applications using the WOSA interface will thus be able to access information resources across multiple computing environments. Total achievement of this goal remains a few years away, but development in certain areas is already underway. WOSA will undergo constant change as it adapts to new technology and user requirements.

Microsoft announced the WOSA initiative in early 1992. Existing products focusing on heterogeneous connectivity, distributed computing, and groupware support were assembled into the initial package. Early entries to the WOSA inventory included a joint Microsoft/Digital Communications Associates Communications Server, a Messaging Application Programming Interface (MAPI), and a module called Open Database Connectivity (ODBC). Additional services currently being developed are centered on security and directory services.

WOSA's operational plan (see Figure 1.1) includes an abstraction layer that provides interaction with heterogeneous computing devices via a set of APIs. Windows-based applications, using these APIs, can operate from a variety of end-user devices. New end-user devices can be added as they enter the marketplace. Meanwhile, applications remain unchanged as long as they employ WOSA APIs.

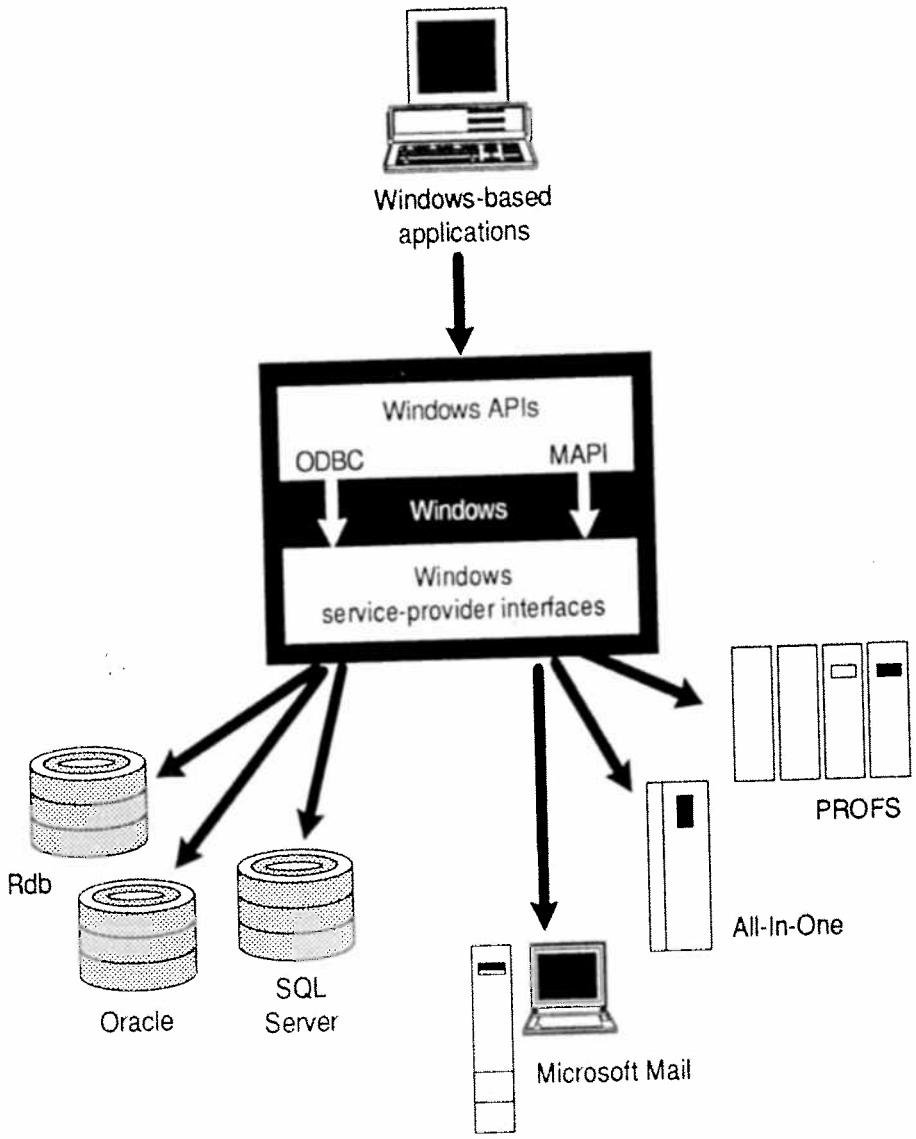
At the service-provider end, additional interfaces link to diverse functional packages. These include numerous database packages, mail utilities, etc. As with the aforementioned end-user APIs, service-provider interfaces (SPIs) can be expanded to encompass new products. Applications will remain unchanged as long as functional packages support the interface conventions defined by its SPI.

Common Interface

The Windows operating environment provides users with a uniform application interface. Once a methodology is learned, it generally applies to all applications. WOSA, in like manner, presents to distributed application programmers a standard interface for interacting with functional packages such as database managers and messaging systems.

Instead of having to learn a different set of APIs for each implementation of a service, programmers creating WOSA applications need only learn a single set of APIs for all implementations of a particular service. In addition, applications remain stable no matter what changes are made to functional services as long as these services communicate through the WOSA interface.

Figure 1.1 WOSA's Operational Plan



Microsoft's goal with WOSA is to control the desktop via Windows. WOSA helps make the Windows operating system a strategic platform for users at various levels of the enterprise. Ultimately, with the release of additional products, Microsoft would like to climb higher into the corporate processing hierarchy. Products such as Windows NT are seen as elements that will facilitate that climb.

WOSA Architecture

WOSA provides a single, consistent, system level interface between Windows-based PCs and various enterprise computing resources (see Figure 1.2). By exploiting the WOSA interface, a Windows-driven desktop application need not know anything about computing resources on the network in order to gain access to enterprise functions such as mail, databases, licensing, or remote procedure calls (RPCs).

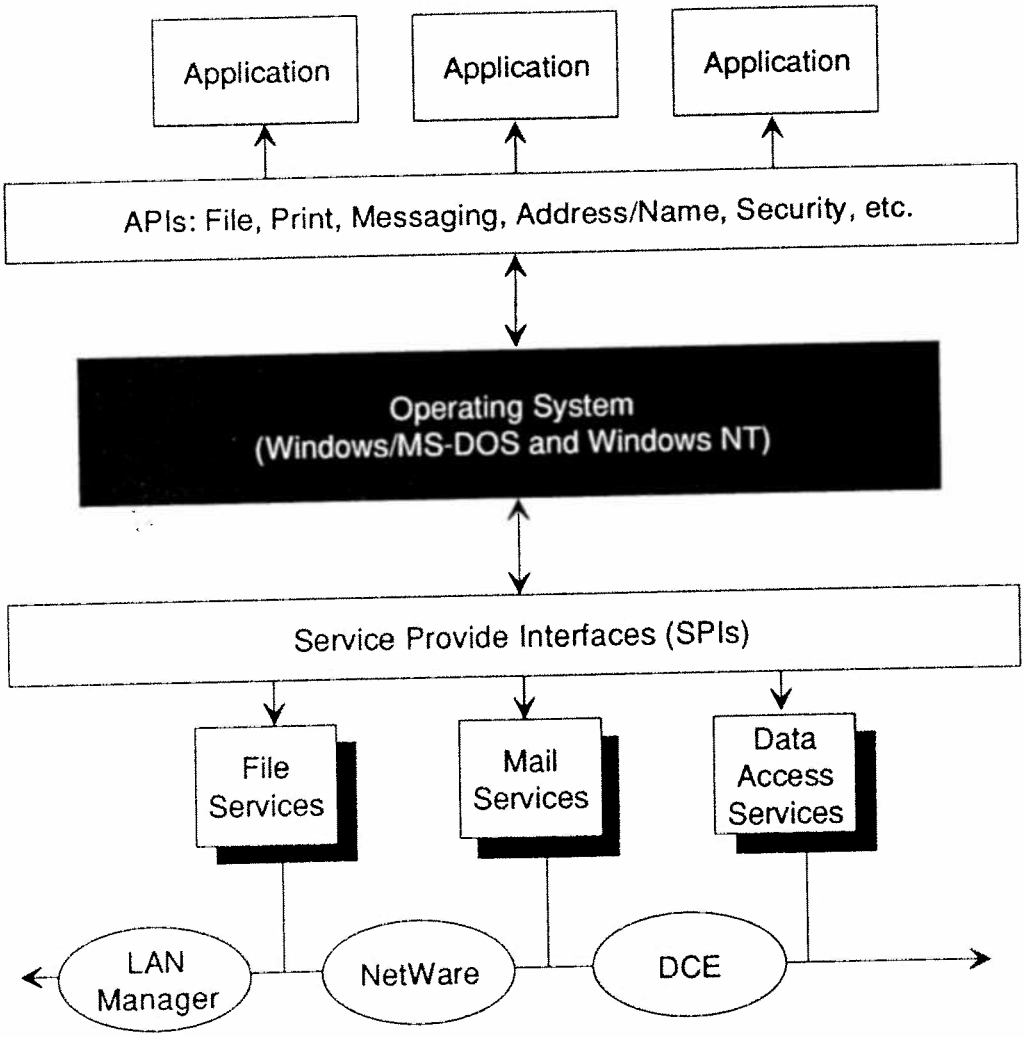
Previously, if an existing functional service such as a database management system (DBMS) was replaced, front-end applications would then have to be altered to accommodate the new service. This is because their API calls to servers were linked in at compile time. Even if the application developer had the necessary resources to write to the new server's API, the existing applications would have to be updated to recognize the change.

WOSA solves this problem by communicating to servers through APIs. They can be linked in at runtime via Windows Dynamic Link Libraries (DLLs). For each functional service, a Driver Manager (MAPI.DLL, for example) makes the connection between the application and appropriate server driver, i.e., SPI.

WOSA Benefits

The primary benefit of WOSA when fully developed will be to provide full access to enterprisewide computing environments for Windows users. There are several additional WOSA benefits that help users maximize returns from their corporate systems. Among them are:

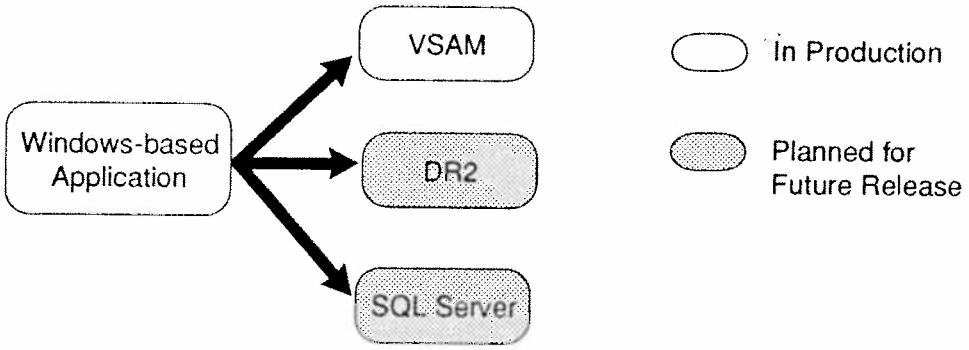
Figure 1.2 Windows Open Services Architecture



- *Easy Upgrade Paths.* Because WOSA enables a single application to work with multiple back-end services, IS managers can upgrade or change those services without affecting the end-users or their applications.

- *Protection of Software Investment.* WOSA protects an organization's software investment by enabling existing applications, without modification, to access new services on a variety of platforms. End-users can work with new resources in the same ways, and using the same applications, as they did with existing implementations (see Figure 1.3).

Figure 1.3 Current and Planned Implementations



- *More Cost-Effective Software Solutions.* As WOSA-based applications become more readily available, IS managers will be able to increasingly use off-the-shelf products to create integrated software solutions that are at least as powerful as more expensive custom alternatives. Moreover, this modular approach makes it easy to tailor software solutions to specific business needs.

- *Flexible Integration of Multiple-Vendor Components.* WOSA's architecture supports multivendor environ-

ments and, in any given environment, multiple implementations of a single type of service. As a result, WOSA makes it easier to switch from one implementation to another. This ability is important for organizations whose long-range plans may require different products than they use today, and it is absolutely critical for companies that are unsure of their long-term requirements.

- *Short Development Time for Solutions.* Creating software solutions for business problems can be a long and expensive process. Solutions based on distributed computing resources can be even more expensive and time-consuming because the complexity of the application is compounded by the need for it to provide access to back-end systems. Since the issues associated with accessing such distributed resources are common to a variety of applications, developers of any given application should not be burdened with the task of resolving problems that are more efficiently and appropriately left to the system software. WOSA relieves developers of this burden by providing a single, open-ended interface for applications at both ends of the network connection.

  By providing access to various implementations of back-end services, WOSA eliminates the need for application developers to develop solutions for each new service implementation. Programmers can provide access to new implementations by plugging existing components together.

- *Extensibility to Include Future Services and Implementations.* WOSA is designed to be extensible, meaning that new types of services can be added to WOSA as needed. WOSA's DLL-based implementation can allow new APIs to be added without disrupting

existing ones. Additional implementations of existing services can be added by building a service provider interface library for the new service provider. Applications can take advantage of new service implementations without being modified and can take advantage of entirely new services only when they need to do so.

WOSA as a Strategic Resource

WOSA's influence on system managers, software developers, and others in the Windows environment can be profound. In addition to its special impact on each group, WOSA also serves as a unifying element among their diverse interests. Some of the technology groups interacting with WOSA include:

- *WOSA and IS Managers.* The typical mission of an IS organization is to provide information services to a spectrum of users whose needs are often quite diverse. In the absence of a truly flexible platform, IS departments must often decide whether to build custom solutions for each type of user or force all users to compromise individual needs by adopting a single common "solution" that fails to serve anybody well.

  WOSA relieves IS managers of the need to make such compromises. Each set of users can use the software that best suits its needs, and IS managers can feel confident that whatever applications are chosen will work together seamlessly. WOSA's open-ended architecture means that IS managers can deploy solutions now without fear of interfering with strategies planned for the future. They know that the mix of front-end applications and back-end services they are currently using will work in the future.

- *WOSA and Systems Integrators.* WOSA simplifies the obstacles that are often associated with integrating

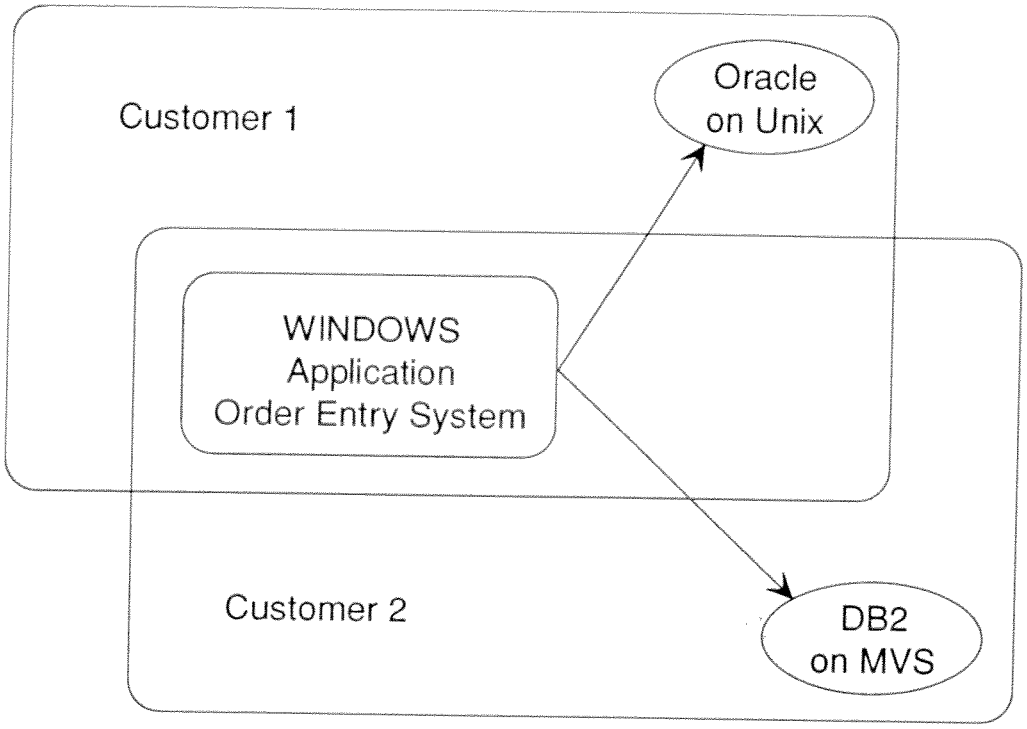
custom software packages into a heterogeneous environment. WOSA's flexibility enables systems integrators to combine custom front-end applications with commercially available software packages and back-end services to build effective solutions.

WOSA reduces the amount of custom software that must be developed to solve a customer's application needs because it allows various pieces of software to be plugged together to form a comprehensive package. WOSA-based technologies enable systems integrators to utilize a single solution in a variety of customer computing environments. For example, an order entry system that works with one customer's Unix-based Oracle server can be used to support another customer's MVS-based database system (see Figure 1.4).

- *WOSA and Corporate Developers.* WOSA helps corporate developers define a stable set of interfaces on which applications can be built, thereby eliminating the need to modify applications in order to access new implementations of distributed services. WOSA also reduces the burden of software support that would be required if a special version of a given front-end application were required for each implementation of a back-end service. WOSA saves development time because programmers can focus on a single set of APIs for each type of service rather than a new set of APIs for each implementation of a service.

- *WOSA and Independent Software Vendors.* Independent Software Developers (ISVs) can gain similar benefits from WOSA as do corporate developers. In addition, ISVs can use WOSA as a way to market a single implementation of their application to multiple service providers. An example is an ISV with a front-end application that accesses back-end database services. In

Figure 1.4 Order Entry Package in Two Different Customer Settings



the absence of a platform like WOSA, the ISV would have to provide explicit support for each type of back-end database engine that the customer base required. Because adding support for new types of database servers is costly, the ISV's potential market might be limited to those organizations that use the database engines already supported by the ISV's product. With WOSA, the ISV's market opportunities can be expanded to include customers who use any database service that supports the WOSA interface.

- *WOSA and Back-End Service Providers.* With over 100 million PCs worldwide, vendors of back-end computing services are faced with the challenge of providing PC connectivity to their products. These vendors must not only provide the system software necessary to access their services from desktop systems, but must motivate

customers and vendors to either build PC-based front ends, or develop and market such packages themselves.

With a standard API to access a variety of back-end services, WOSA reduces the burden on the back-end service vendor in both of these areas. Because new implementations of back-end services can be accessed from a common API, the vendor's system software requirement is limited to providing the service provider interface library for their service. Applications that use WOSA to integrate services from various back-ends can be used without modification with new implementations of the service.

Microsoft Windows

Windows is the center of Microsoft's system software strategy. Windows implementations fall into two broad categories. One category has Windows running with current and future versions of the MS-DOS operating system. This implementation has been steadily upgraded to exploit the Intel 80x86 processor series, and includes features needed for a majority of desktop users. It has retained its role of being a direct complement to MS-DOS through successive releases.

Another Windows category is manifested in the Windows NT operating system. The latter offers advanced operating system features needed for more demanding desktop applications, including high-performance server routines used in client/server configurations and downsized applications offloaded from host-based systems.

The first Windows category, i.e. DOS-based, is targeted for the Intel platform. Its ideal environment is the 80386 processor and up, although it will operate with reduced performance on the 80286.

Windows NT is geared for more demanding tasks. It can handle large, resource-intensive jobs in conjunction with powerful hardware support. A full 32-bit system, NT provides features such as security, multiprocessor operation, and is portable to Reduced Instruction Set

Computer (RISC) and other platforms. The operating system is also POSIX-compliant. POSIX (Portable Operating System Interface) is an operating system interface standard that enhances application portability.

The broad scalability of Windows is depicted in Figure 1.5. Applications based on Windows can operate on a wide range of computing systems.

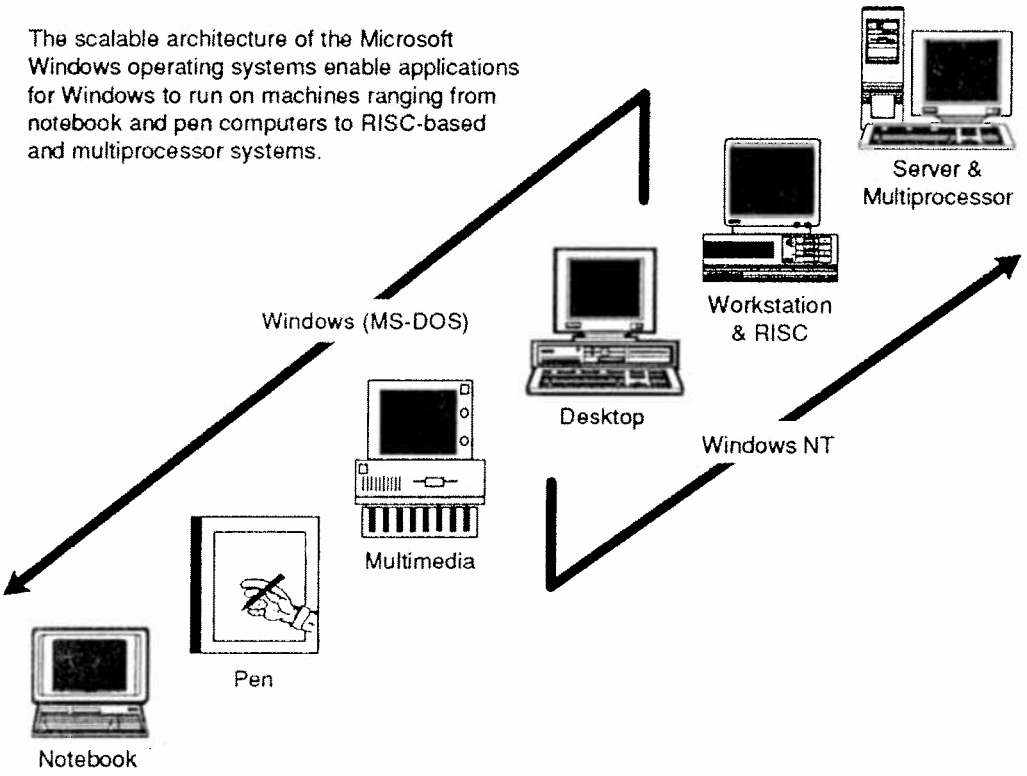
The most popular release of Windows so far has been 3.x. With its advanced Setup program, built-in tutorials, and uniform drop-down menus, this Windows version achieved widespread acceptance. Improvements in memory use, disk access, and printing speed gave 3.0x a large advantage over earlier releases.

New capabilities added to version 3.1 include TrueType scalable fonts which generate high-quality output on screen and printer. An object linking and embedding (OLE) protocol supports integration of information from separate applications. Full multimedia capabilities have also been built into 3.1. This makes it possible to create and manipulate digital audio through the Windows interface, and control multimedia devices such as digital video cards and videodisc players. The OLE protocol allows users to embed a multimedia entity, such as an audio or video clip, into an existing Windows application, just as they would a chart or text file.

Microsoft Windows 3.1 is also the initial platform for extensions that allow users to exploit computing technology such as pen-based computers. To leverage the potential of pen computing, Microsoft has developed Windows for Pen Computing, which supports the use of a stylus input device in place of a mouse and keyboard. Other features include high-quality handwriting and shape recognition and a pen message interpreter that enables existing applications for Windows (and MS-DOS) to use the pen. More than 20 hardware vendors now ship Microsoft Windows for Pen Computing with their hardware.

The Windows operating system is optimized for network performance, with features such as automatic network reconnection and interfaces

Figure 1.5 Windows Scalable Architecture



tuned for interoperation with many of the leading network operating systems. Microsoft offers full Windows-based networking with the LAN Manager family of network computing products. Microsoft LAN Manager is supported on OS/2, Unix, VMS, and Windows NT platforms.

Windows NT represents an advance in operating system functionality for networked PCs and network servers. It possesses built-in peer-to-peer networking. Windows NT also supports networking control capabilities, including network monitoring tools, centralized workstation administration, and performance management tools. The underlying Windows NT operating system is specifically designed for high-speed network operations, with a new file system, high-speed drivers, and enhancements in other performance-sensitive areas. Windows NT also will support Microsoft's current server applications, including Microsoft SQL Server and DCA/Microsoft Communications Server, and server

applications from other vendors. Microsoft has articulated plans to continue to deliver and support LAN Manager, SQL Server and Comm Server on the OS/2 platform. Microsoft will also provide a migration path for those customers interested in moving their servers from OS/2 to Windows NT.

Table 1.1 portrays the evolution of Windows networking from both client and server perspectives.

Table 1.1 Evolution of Networking with Windows and Microsoft LAN Manager

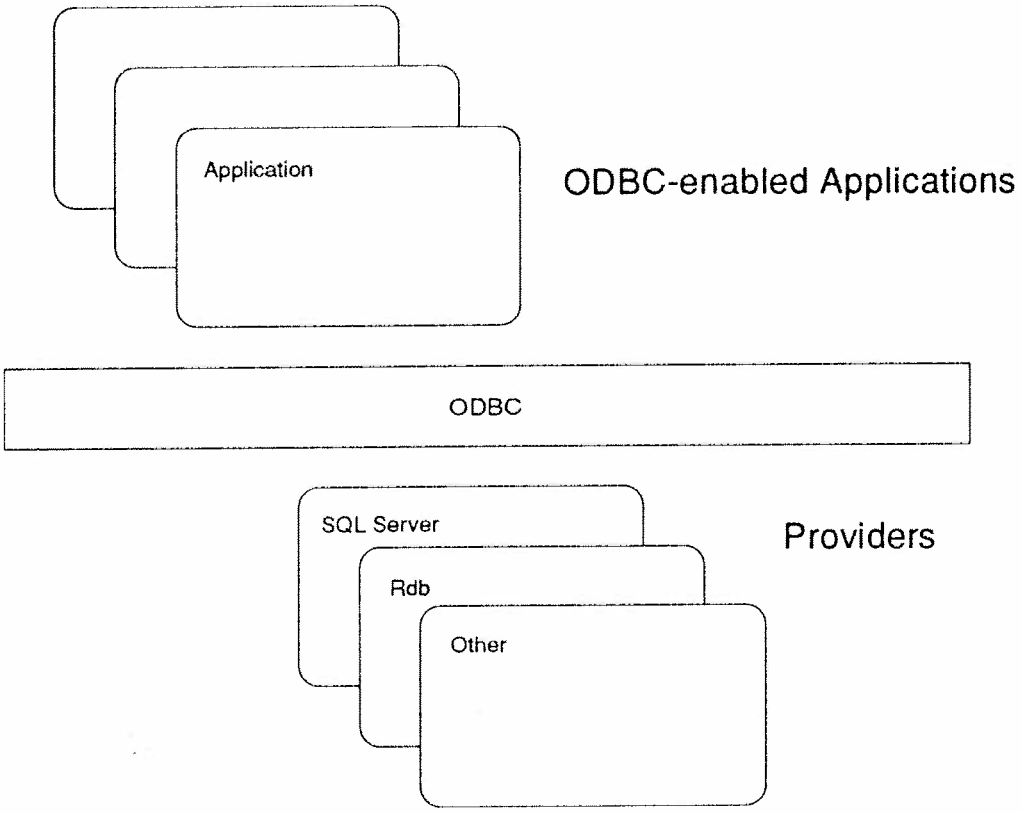
	Today	Short Term	Future
What the CLIENT runs	• Windows 3.1 (MS-DOS)	• Windows 3.1 (MS-DOS) • Windows NT	• Next release of Windows (MS-DOS) • Next release of Windows NT
What the SERVER runs	• LAN Manager for OS/2 • LAN Manager for UNIX • LAN Manager for VMS (Pathworks)	• LAN Manager for Windows NT • LAN Manager for OS/2 • LAN Manager for UNIX • LAN Manager for VMS (Pathworks)	

Open Database Connectivity

ODBC is the database component of WOSA. It is a strategic interface for accessing data in a heterogeneous environment of relational and non-relational database management systems (see Figure 1.6). Based on a call-level interface (CLI) developed by an industry consortium of more than 40 vendors (the SQL Access Group), ODBC defines a common API for accessing heterogeneous database information in a uniform fashion. With ODBC, application developers can allow an application to concurrently access, view, and modify data from multiple, diverse databases. Apple Computer has endorsed ODBC as a key enabling technology for retrieving data from System 7 applications, and will integrate ODBC support into System 7. With growing industry support, ODBC is emerging as an important industry standard for data access for both Windows and Macintosh-based applications.

ODBC provides a common data access API. Each application uses the same function calls to talk to many types of data sources through

Figure 1.6 ODBC Architecture



DBMS-specific drivers. A driver manager sits between the applications and the drivers, just like a print manager sits between applications and print drivers. In the Windows operating system, the driver manager and the drivers are implemented as DLLs. Through the ODBC API, applications can do the following:

- Establish and drop connections with remote databases.
- Present a standard logon interface to users.
- Find out what tables, views, columns and indexes are available on the remote database.
- Issue commands and retrieve result tables.

- Open fully scrollable cursors for record-at-a-time processing.

- Receive detailed, standard error messages based on an ANSI SQL Error specification.

The ODBC driver manager loads drivers dynamically as they are needed. The driver, developed separately from the application, sits between the application and the network. The driver processes ODBC function calls and translates them to the commands required by the target data source.

ODBC has received widespread industry acceptance, with many endorsements from within the database vendor community, as well as from application vendors and corporate developers. ODBC's acceptance to date is due to several factors.

- As an implementation of the SQL Access Group's CLI specification, ODBC is vendor-neutral and open. This open systems approach solves a problem common to everyone in the software industry.

- As a portable API, it can be a common data access language for both the Microsoft Windows operating system and the Apple Macintosh operating system, and possibly other operating systems.

- ODBC allows application developers to decide for themselves when to optimize their ODBC implementations for maximum interoperability, or fully exploit advanced features of a particular DBMS.

Distributed Relational Database Architecture (DRDA) is IBM's protocol for connecting databases in the IBM Information Warehouse over Systems Network Architecture (SNA) networks. ODBC does not compete with DRDA. Rather, it accesses DRDA-compliant databases through

ODBC drivers in the same manner that it accesses other database servers. ODBC drivers for IBM databases are under development by such companies as Micro Decisionware for DB2 and Rochester Software for SQL/400.

ODBC is not a product that users buy. It consists of: a Software Development Kit (SDK) for application vendors wishing to take advantage of ODBC connectivity; a driver manager available from Microsoft; and individual server drivers, available from Microsoft or the server vendor, depending on the driver. ODBC is currently available as a component of the Microsoft Access DBMS. The ODBC driver manager will also be a standard feature of the Windows operating system.

Messaging API

MAPI first appeared as a Windows 3.x subsystem. It is gradually being ported to other platforms such as Windows NT, OS/2, Macintosh, and MS-DOS. Currently, however, MAPI is optimized for the Windows platform, i.e. user interface, memory management, file formats, etc. In this sense, it remains a proprietary messaging vehicle.

MAPI has been designed to assure developers who write front-end applications or back-end services for their operating environments that they will interoperate successfully. This is done by publishing the interfaces between these services in the form of APIs and SPIs (see Figure 1.1).

All of the functionality required for E-Mail front-ends is delivered in MAPI by a set of calls to transport, message store, and directory services. MAPI provides access to multiple transports and message stores. This access is achieved via DLLs written to the MAPI SPI.

Windows platforms supporting MAPI have initially shipped with the client mail application entitled Mail Manager. The latter is directly accessible to the user or becomes available when a mail-enabled application, such as a spreadsheet or word processor, is activated.

MAPI supports a wide range of message types and appendages, including OLE attachments. OLE attachments allow a user to access the native application of an embedded object for the purpose of viewing and editing. MAPI does not, however, track Dynamic Data Exchange (DDE) links among objects once one of the objects has been transmitted.

The MAPI architecture, as with other WOSA services, provides client APIs at the user end and SPIs at the service provider end for various heterogeneous products. Both APIs and SPIs are written to the Windows messaging subsystem entitled Mail Spooler. The latter is a DLL that is an inherent part of the Windows operating system. It also provides a portion of the functionality, in concert with the API and SPI, that constitutes the MAPI service model.

There are two MAPI versions: Simple and Extended. Simple MAPI is suitable for sending and receiving interpersonal messages, including attachments. Application developers will typically use it to send and receive messages from within their own applications.

Extended MAPI allows applications to manage the creation and transmission of more complex forms of messaging and addressing. These applications span a range from data collection to agent-based retrieval to other more esoteric forms of message-generating system entities. Extended MAPI also offers advanced addressing and messaging management.

MAPI employs object-oriented programming methods. Messages, attachments, etc. are created in the object mode. They invoke the object characteristics of polymorphism (two or more objects allow the same calls) and inheritance among objects that facilitates the development and maintenance of MAPI-based applications.

Microsoft views Simple MAPI as harmonious with the status of VIM (Vendor-Independent Messaging) promoted by Lotus and others, at least in terms of its cross-platform potential. Whether MAPI can escape its close ties to Windows while retaining full functionality is an issue yet to be determined.

License Service API (LSAPI)

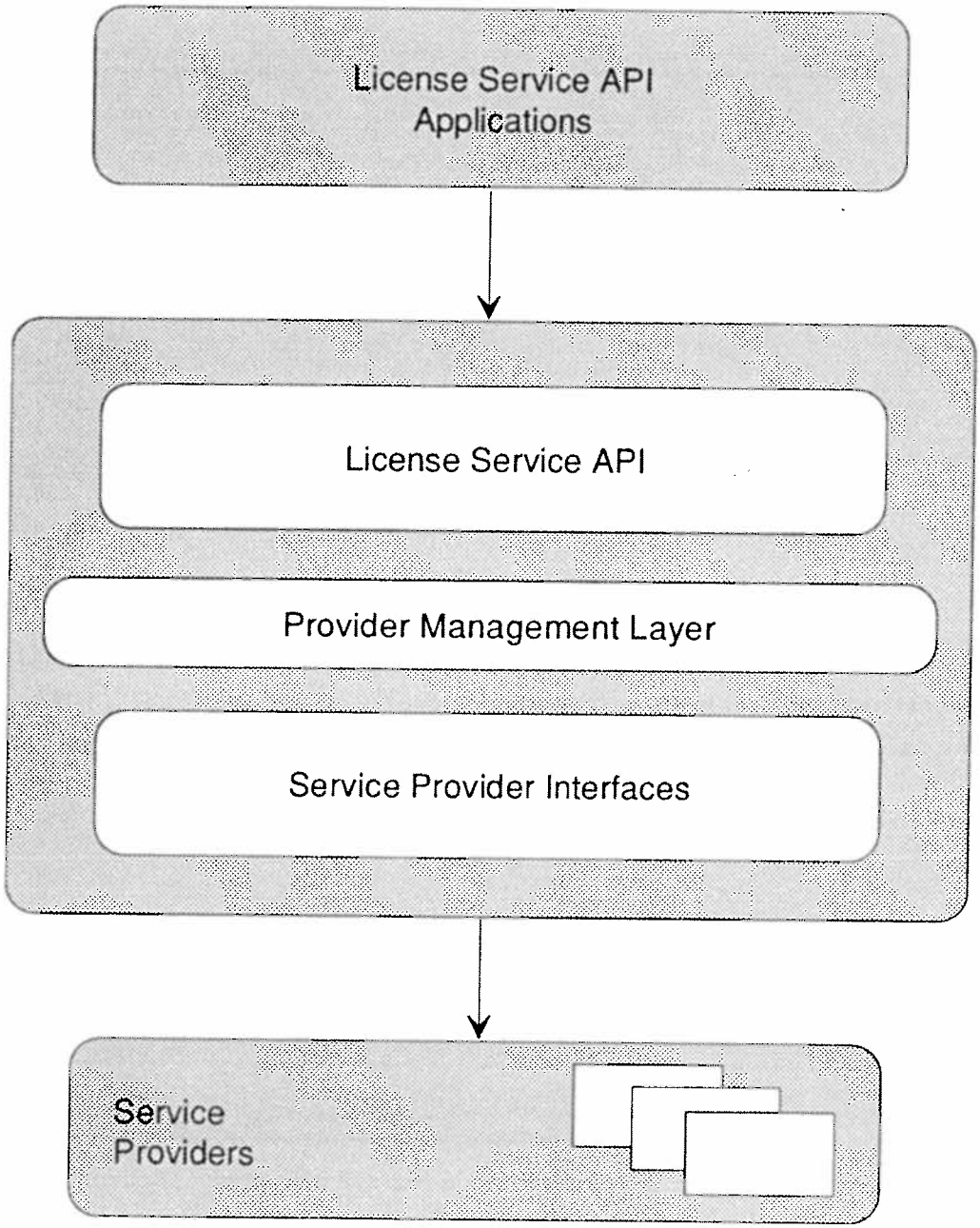
The LSAPI is a specification that allows software publishers to build licensing capabilities into their applications while remaining independent of any specific licensing system or model. This feature offers IS managers the flexibility to plug in new licensing services without having to modify applications on their client PCs.

WOSA enables LSAPI-compliant applications to interact with licensing services transparently in a multivendor networked environment (see Figure 1.7). Thus, applications can access licensing systems over a variety of networks, easing the development process for programmers and increasing installation options for IS managers. Over 20 software and system vendors have proclaimed their intention to support the LSAPI specification, with more being added on a regular basis. The problem addressed by LSAPI began in earnest with the arrival of the first PCs into the corporate mainstream. System administrators and product vendors have been trying to control and manage the use of software applications in this setting ever since, with varying degrees of success.

Typically, to help control software usage, vendors ask purchasers to sign a license agreement and then enforce the agreement through a number of different procedures. However, as the number of license agreements grows, the task of ensuring compliance with the variety of terms and conditions becomes overwhelming. To ease the burden, a number of software products have emerged that track application usage, and some of these can electronically enforce the terms of the license agreement. Some software publishers incorporate their own license tracking mechanisms directly into their applications without relying on third-party license software.

The result is a complex proliferation of different license systems, each with its own non-standard API. The large number of APIs creates higher development and maintenance costs for software publishers since they must write new code whenever they want to interface to a different license system. Likewise, the myriad licensing schemes and applications

Figure 1.7 License Service API Implementation Within WOSA



support for each scheme confuse information systems departments and make license management more difficult.

With the License Service API, an application does not need to know anything about the type of network in use, the types of computers in the enterprise, or the types of license policies available in order to enjoy seamless access to a licensing mechanism. As a result, even if the network, computers or license policies change, the desktop application does not need to be rewritten. In other words, the License Service API enables applications to connect to the licensing services they need across multiple computing environments in a platform-independent manner.

The LSAPI is not a product of Microsoft or any other software vendor. It is a specification that has been developed by a group of software publishers and systems vendors for the purpose of standardizing the interface to licensing services. The License Service API effort was founded by Brightwork Development, DEC, Gradient Technologies, Microsoft and Novell. Many other vendors have subsequently become involved in the effort.

Windows Sockets API

Developed by the University of California for its Berkeley Software Distribution (BSD) version of Unix, sockets are a widely-used programming mechanism that supports data exchange over a network. A socket is a connection point through which data can be sent and received. Originally used exclusively by Unix applications for communications over TCP/IP networks, a variety of sockets implementations have appeared which function in non-Unix environments to allow information transfer between Unix applications and their non-Unix respondents.

Window Sockets encompasses both familiar BSD socket routines, along with extensions specific to a Windows environment. It is intended that Windows Sockets provide a single API to which application developers can program and network software producers can conform. Ultimately, protocol stacks other than TCP/IP will be supported.

This API is targeted for use with all versions of the Windows operating system from Release 3.0 and up. It provides, therefore, for applications which function in both 16- and 32-bit operating environments.

There are two types of sockets currently available to users: stream and datagram sockets. A stream socket supports a two-way, reliable, sequenced, and unduplicated data flow without record boundaries. A datagram socket provides for a two-way data flow which is not guaranteed to be reliable, sequenced, or unduplicated, but which does preserve record boundaries. The stream variety is essentially connection-oriented, whereas a datagram socket is connectionless. The latter type resembles the approach found in many packet-switched networks, of which Ethernet is a prime example.

By supplying a single interface through which applications for Windows can communicate with Berkeley sockets-based applications and services on Unix platforms, the Windows Sockets API relieves programmers from the burden of supporting multiple sockets APIs within their applications for Windows. Without a standard sockets interface in the Windows environment, programmers developing applications for Windows would be forced to modify their code to supply different vendors' Windows-based TCP/IP products.

The addition of Windows Sockets to WOSA broadens the WOSA framework by enabling seamless communication between the Unix and Windows environments. Windows Sockets provides a programming solution to a wide range of Windows to Unix communication requirements, including Windows-based client to Unix client, Windows-based client to Unix server, and Windows-based server to Unix client.

Windows Sockets also supports several extensions that allow applications to take advantage of features in the Windows environment. For example, a Unix-based application can interface with Windows through a network and take advantage of such Windows features as multitasking. Support for multithreaded processes is also included. Microsoft is also currently implementing the Windows Sockets API in the Windows NT operating system.

Windows SNA API

This specification defines a standard interface between applications for Windows and IBM System Network Architecture (SNA) protocols. An application written to these standard interfaces will run without alteration over many vendors' SNA connectivity products. This applies to any 3.x version of Windows, as well as Windows NT.

Table 1.2 shows over 20 vendors who collaborated in the development of Windows SNA API.

Table 1.2 Vendors Involved in Developing Windows SNA API

- Andrew Corporation
- Attachmate
- Computer Logics
- Data Connection
- Digital Communications Associates
- Easel
- Eicon Technology
- FutureSoft
- IBM
- ICOT
- International Computers Ltd.
- Microsoft
- Multi Soft
- NCR
- Network Software Associates
- Novell
- Olivetti
- Siemens-Nixdorf
- Systems Strategies
- Wall Data

Developed by IBM, SNA is the pre-eminent proprietary network model for data interchange. The SNA protocol set was designed exclusively for use in mainframe-based environments in its early years, but now has assumed a more open, peer-to-peer stance. SNA has essentially served as the inspiration for all network-layered configurations in use today.

The growth in the personal computer market has created opportunities for vendors to produce and market PC-to-mainframe connectivity solutions. These solutions offer users a means to access data on a mainframe by allowing them, for example, to emulate 3270 terminals on their PCs. Many of these vendors also have offered their own APIs that allow developers to build applications on PCs that exchange data with mainframes. Now, through this cooperative effort, a standard Windows SNA specification has been defined.

Microsoft's rationale for support of the SNA API is to bolster Windows-to-host connectivity. In addition, this provides the company with yet more leverage in its bid to extend its influence into the enterprise environment.

Prior to this standard-setting endeavor, each vendor offering PC connectivity solutions for the SNA market created its own unique interface. This made it impractical for an application to use different connectivity products without major change to its own interface connections.

The Windows SNA standard covers all five of the SNA capabilities that programmers use today. These are High Level Language API (HLLAPI), Common Programming Interface-Communications (CPI-C), Advanced Program-to-Program Communications (APPC), Logical Unit (LU) 0, and Common Service Verbs (CSV).

The HLLAPI interface is used with existing IBM 3270 and 5250-based applications. Both the APPC and CPI-C APIs are used to write cooperative applications for the LU6.2 protocol. The LU 0 API is used to gain access to low-level SNA LU 6.2 data streams that are frequently encountered, particularly in financial environments. The CSV API performs character set translations and interfaces with IBM's NetView management package.

Despite IBM and Microsoft antipathy in the operating system arena, they, along with their collaborators, worked harmoniously to develop the SNA API. It was a case where each of their mutual interests were served – Microsoft to penetrate the enterprise and IBM to extend the SNA environment to a wider user community.

Remote Procedure Call API

Windows' RPC falls into the communication service category of WOSA offerings. Along with Windows Sockets API and Windows SNA API, it provides services dealing with remote access.

RPC is closely aligned with the client/server model. This is where the client typically manages the end-user interface while the server deals

with database management, system services, and special purpose processing needs.

RPC is also a tool which enables the single PC to extend its reach far beyond its own domain. By enabling access to all computing resources on the network, RPC provides users with a powerful tool for completing tasks and producing solutions.

Microsoft RPC, as with other varieties, makes a remote access procedure appear to be a conventional local procedure call. In actuality, it is made to a "stub" that interacts with the runtime library and performs all of the steps required to execute the call in the designated remote environment.

The RPC model is an industry standard, although there are variations in implementation. Microsoft's version is similar, although not identical, to that put forth by the OSF in its DCE package. Sun Microsystems' RPC is yet another popular version.

RPC also renders network interface details transparent to the application developer. No longer must he or she understand specific network API functions or low-level communications protocols in order to build complex distributed applications.

Data translation problems that frequently arise during interaction among heterogeneous networks is reduced by RPC usage. Applications can now ignore issues of formatting and character structuring, all of which is handled during RPC execution.

WOSA Extensions For Financial Services

In 1992, Microsoft formed a consortium of firms interested in financial services markets in order to standardize the end-user interface. Among these organizations were computer vendors, software houses, and system integrators. Labeled the Banking Systems Vendor Council, early members included Microsoft, Unisys, Olivetti, DEC, and Andersen Consulting.

Their basic goal was to allow any application using Windows to employ standard interfaces for access to financial data and devices. The specification is intended to be usable within all versions of the Windows operating system, from 3.1 and up to Windows for Workgroups. Initial versions of Windows NT are also acceptable environments. Thus, the API can be used in both 16- and 32-bit configurations.

Although WOSA Extensions for Financial Services (XFS) provides a broad architecture (see Figure 1.8) for accessing service providers from Windows applications, initial tasks have been focused on supporting an interface to user devices that are peculiar to financial institutions. These devices are often unique to their function, proprietary in nature, and difficult to support from a general purpose operating system platform. Solving these complex linkage problems, therefore, provides immediate relief to the financial community.

Issues to be dealt with in the near future include:

- financial transaction management and message control
- security
- network and system management

In the longer term, the consortium will address promising new technologies such as multimedia processing, object-oriented development, pen-based computing, and wireless computing.

Future Directions

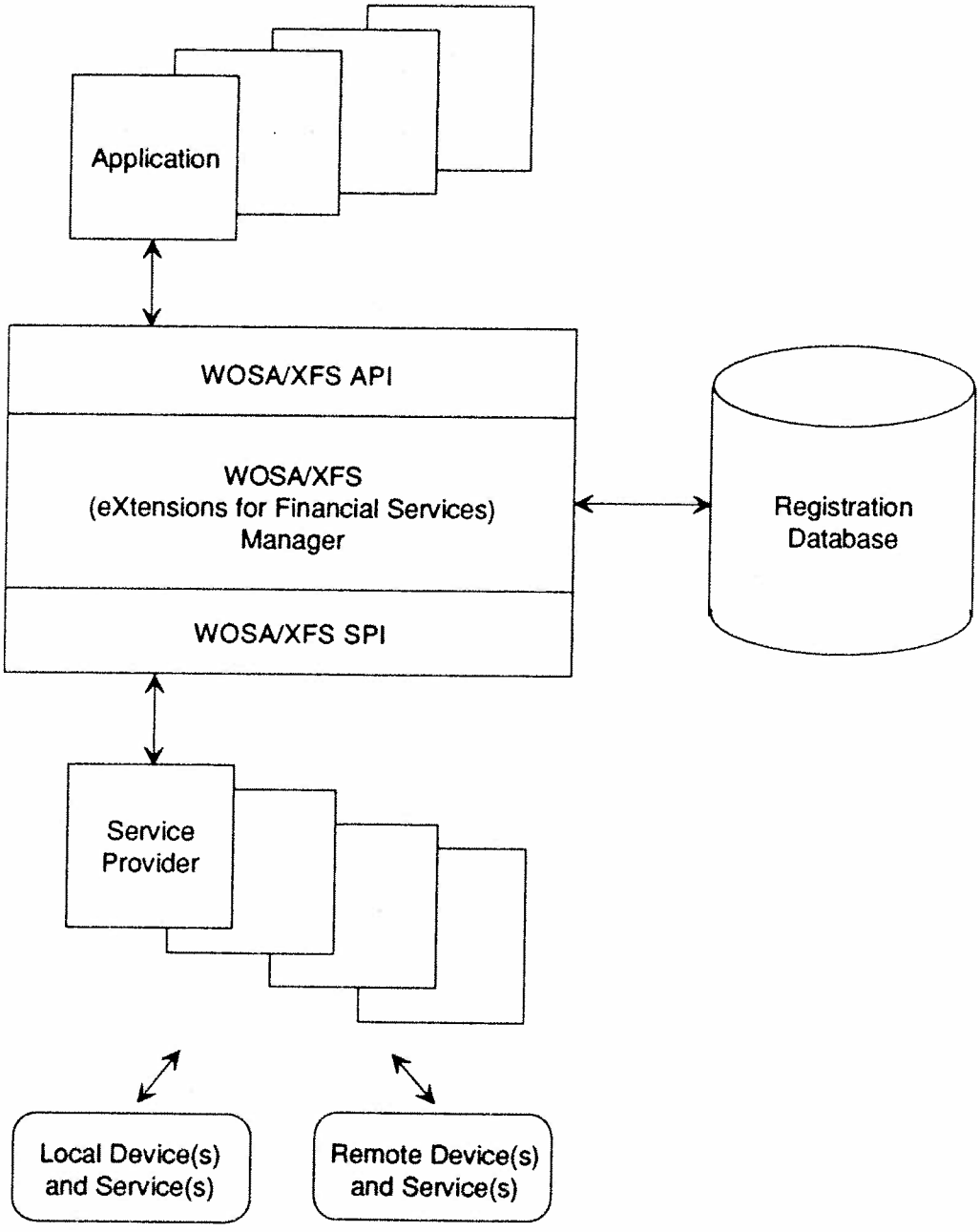
WOSA is designed to be extensible. New capabilities will be added over time. Among these will be services dealing with systems management, distributed file systems, directory services, wider communication options, security in distributed environments, etc.

WOSA's DLL-based architecture enables new APIs to be incorporated without disturbing current capabilities. New varieties of existing services can be added in a straightforward manner by creating an SPI library for the emerging product. Applications are unaffected by any of these additions or modifications.

Microsoft plans to continue to extend the Windows operating system in the future. The system is being enhanced using object-oriented techniques to yield improved usability, a more intelligent file system, a more intuitive user interface, and more transparent operation in connected environments. Users will be able to browse network resources using a variety of object attributes such as content, and creation and revision dates. Both Windows and Windows NT implementations will benefit from these enhancements.

Windows running in enhanced mode with MS-DOS will be extended to support the Windows 32-bit API. As part of this evolution, Microsoft will enhance MS-DOS over time, reduce memory requirements, add utilities, and improve networking support. In this way, the MS-DOS-based implementation of Windows running in enhanced mode will be further optimized for laptops and desktops, while Windows NT will be Microsoft's solution for high-end machines such as workstations and servers.

Figure 1.8 WOSA Extensions for Financial Services



Chapter 2

The Windows Environment

The appeal of Windows lies in its ease of use and universality. Even novice users can access sophisticated programs and services. The learning curve is moderated by graphical interfaces. Common tasks such as file management and printing have been simplified.

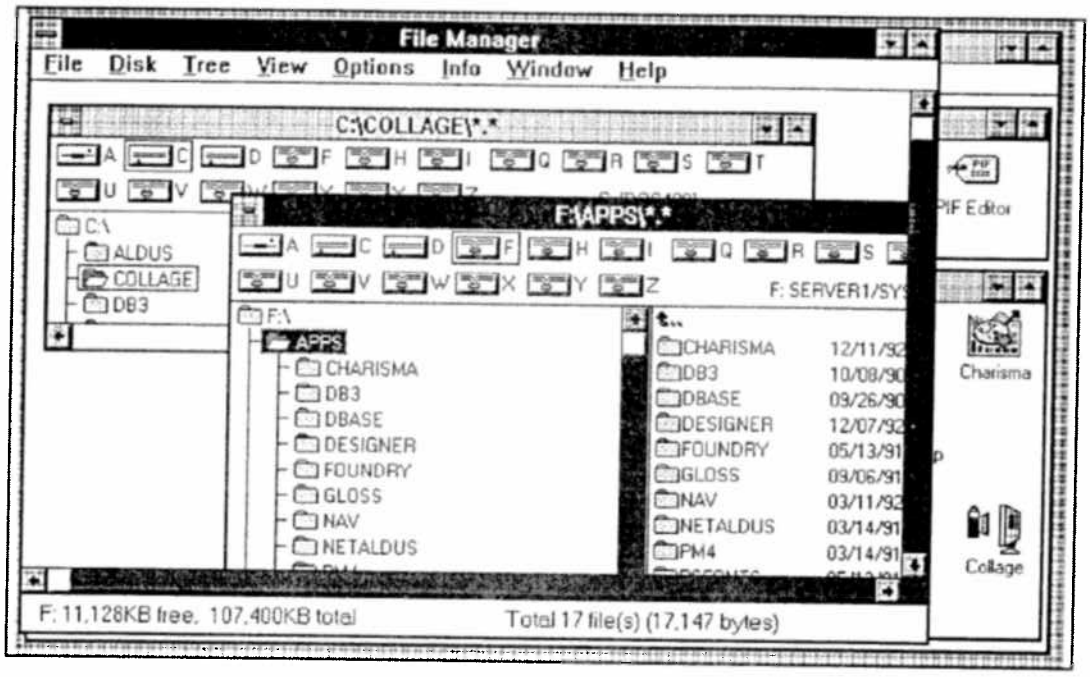
Among the graphical tools available are:

- *Program Manager.* Manipulates and arranges icons in order to implement applications.
- *File Manager.* Manages files and their associated directories to support user access, manipulation, and consolidation of information repositories (see Figure 2.1).
- *Print Manager.* Allows users to view print queues, change the status of print jobs, and direct output to specific printers.
- *Control Panel.* Sets up communications ports, installs fonts, and allows customizing of visual images in terms of colors and designs.

Among the desktop applications made available are:

- *Terminal.* Asynchronous terminal emulation and communications support program that includes a macro capability for up to 32 function keys.
- *Calendar.* Builds a daily appointment schedule and monthly calendar which initiates alarms as appointment reminders.

Figure 2.1 The Windows File Manager Screen



- *Calculator.* Features both standard and scientific calculators that perform calculations in decimal, binary, octal, and other notations.
- *Recorder.* Captures mouse actions and keyboard strokes in macro form for subsequent reuse.
- *Clock Displays.* A clock in digital or analog format anywhere on the screen.
- *Cardfile.* Presents an index card-oriented filing program which can store both graphics and text.
- *Notepad.* A simplified text editor supports notes, memos, and batch files with date and time stamping.

Perhaps the key advantage of using Windows, over and above its facile interactions, is the multiple thousands of applications available for use. Third-party software developers have migrated to the Windows environment in droves. End-users are the ultimate beneficiaries of this widespread support.

Performance and Reliability

Successive releases of Windows have witnessed continued performance improvements. Windows 3.1, for example, offers the following enhancements over Release 3.0:

- Faster, more responsive user shell components (notably File Manager and Program Manager).
- Faster disk caching. The Windows SmartDrive disk caching utility was redesigned for Windows 3.1. It is installed automatically during setup and boosts performance by caching read and write disk operations.
- Faster paging in 386 enhanced mode. Windows 3.1 includes 32-bit disk access which allows the operating system to bypass the MS-DOS operating system for its virtual memory paging file.
- Increased display driver performance.
- Better printing performance. Overall speed is improved, and Windows 3.1 gives control back to the application quickly once the print command is invoked.

Table 2.1 shows a few examples of speed increases that took effect with the release of Windows 3.1.

Table 2.1 Examples of Windows 3.1 Speed Increases (in Seconds)

Task	Windows 3.0	Windows 3.1
Move 64 files	32.0	3.2
Save PageMaker file	9.5	6.1
Print Microsoft Excel spreadsheet	30.2	15.7
Page down PowerPoint	18.9	13.9

In Windows 3.0, if a serious application error occurred, the Windows environment reported the error as an unrecoverable application error

and closed the application – sometimes leaving the user with little information about what triggered the problem. To protect the system from the effects of the error, it was recommended that the user exit the Windows program and reboot the system.

Several changes were implemented in Windows 3.1 to improve Windows' reliability and to help software developers improve the integrity of their Windows-based applications. Specific actions taken included:

- protecting Windows itself from application errors through parameter validation;
- providing more information and procedures to help users recover from application errors if they occurred;
- providing users with application reboot options, helping to prevent a system crash or loss of data during the worst of application errors;
- providing test tools and test support to help independent software vendors develop more stable Windows-based applications.

Windows 3.1-based applications are more reliable because of an internal feature called "parameter validation." Without parameter validation, a Windows-based application might make a request from the Windows operating system, such as seeking an allocation of resources, that is incorrect for that application. Windows 3.0 would attempt to execute such requests, perhaps writing over important system data in memory or sending data to invalid addresses. In Windows 3.0 such application errors sometimes caused unpredictable application behavior, data loss, or a system crash.

Windows 3.1 performs an internal check to validate that specific parameters are valid for resource allocation, handles, and pointers that a Windows-based application might request. This check is performed with minimal impact on overall system performance. Incorrect parameters are returned to the application as a failed request, and the application must reprocess the request.

In Windows 3.1, if a user experiences an application error or general-protection fault, the Windows operating system:

- identifies the source of the error.
- provides the user with the option of closing the application or returning to the application, so work can be saved before closing (if the error was detected as being harmless);
- attempts to keep the system running after the error.

If a Windows-based application is malfunctioning, the user can press CTRL+ALT+DEL to close the problem applications without closing Windows. An error message appears with instructions for closing the application, returning to the system, or rebooting the system.

If CTRL+ALT+DEL is pressed when no application is hung, Windows sends a warning that advises the user to quit the application with the correct Quit or Exit command, or by choosing the Close command from the Control menu. This helps guard against losing data when the user's system is actually functioning correctly.

Windows 3.1 went through one of the largest beta programs ever conducted, involving more than 15,000 participants. Microsoft gave developers in the beta programs the SDK and Driver Development Kit (DDK) to allow them to implement Windows 3.1 APIs for their new programs. A Hardware Compatibility Program included over 350 testers to ensure compatibility of Windows with a vast array of hardware and peripherals on the market today.

Object Linking and Embedding

An important technology for many reasons, OLE creates an environment in which applications can share information seamlessly. With OLE, all data can be thought of in terms of objects. A spreadsheet chart, an illustration, a table, a paragraph of text are all examples of objects. OLE enables applications to share these objects.

Windows 3.1 supports OLE by providing standard libraries, interfaces, and protocols that applications use to exchange data objects. As developers implement OLE capabilities within programs, users will encounter a new generation of applications in the market that are designed to work together.

Microsoft added OLE capabilities within new versions of the Write, Paint, and Cardfile accessories that are provided with 3.1. A user can, for example, create an illustration using the Paint program and embed the graphic in a Write document. If the illustration must be updated, the user can re-access the image within the Write document, which launches Paint automatically to allow editing of the drawing. Because the original graphics file is embedded in the document, there is no need to store or update multiple copies of the image, and the file can be updated on any PC with Paintbrush.

OLE is an enabling technology for multimedia computing. With Windows 3.1, a user can embed a multimedia object, such as an audio clip, into an application for the Windows operating system, just as they would embed a chart or text file. The extensible architecture of the Windows operating system makes it possible for multimedia computing to range from low-cost systems to sophisticated multimedia platforms at the high end of the market.

Windows for Pens

Pens offer unique advantages. They are familiar and natural input devices. They adapt well to small forms and can be used when either standing up or sitting down. The pen is also a basic graphical instrument with which users can print characters or draw free-form images and symbols. PCs are evolving to smaller, more adaptable forms that look more like clipboards, notepads, and pocket calendars than desktop systems. Electronic clipboards will enable new categories of users (including package delivery people, inventory pickers, packers in warehouses, and sales representatives) to tap the power of the computer. By incorporating the pen, laptop computers can support note taking, better time management, and more convenient computing in group settings.

Based on Windows 3.1, Microsoft Windows for Pens is an original equipment manufacturer (OEM) product that is packaged and licensed with a variety of third-party pen computers. With Windows for Pens, users can integrate the pen with existing applications for the Windows operating system and development tools. They can point and click as they would with a mouse, issue commands using gestures (for example, draw a line through a word and it disappears), and write documents.

Windows Networking

User demand for networked applications has risen dramatically in recent years. Prior to Release 3.1, however, Windows was deficient in network interface procedures. Three areas where 3.1 and later versions improved over earlier Windows' releases are: network setup assistance, network management, and the addition of a File Manager system with persistent drive connections.

A command new to 3.1 is SETUP/A, which places all files from the original Windows distribution disks onto a network server drive specified by the implementor. This step consumes about 16 megabytes (MB) of server disk capacity, but frees up hard drive space on individual PCs where this data had traditionally been stored.

A SETUP/N command gives each networked PC its own Windows preference files. This is achieved by making the Windows network directory the current directory on a PC, and executing SETUP/N.

In addition, a user's initialization files, e.g. WIN.INI and SYSTEM.INI, which describe hardware configurations, can be stored on a network server. This facilitates future updates to various installation configurations. Rather than updating numerous individual PC files, a network manager need only update a server file. This eliminates the need to go to each PC, some of which may be widely dispersed.

Another positive aspect of storing initialization files on a server rather than on individual PCs is consistency of service. Various types of users can access network services at different locations and obtain the same support they receive at their own PC. A network manager can access

directories anywhere, for example, while a knowledge worker can similarly use a spreadsheet.

The Program Manager package within Windows allows network managers to customize the images viewed by users when they are selecting applications to run. Since initiating a program typically involves clicking a mouse pointing at a visual icon, the total number of icons can reach unmanageable limits in a busy installation. Program Manager overcomes this problem by allowing specific images to appear to specific users, i.e. each user group sees only a subset of the total icons in use, depending on what is needed relative to their work responsibilities.

In practice, this is accomplished by storing within Windows initialization files the specific icon group associated with each class of users. Then, when a particular group logs on to the network, they enter their class identity. This automatically links them to a specific icon group.

File Manager in 3.1 and up has qualities more suited to network operations than was available in earlier Windows versions. One of its more important attributes in a network sense is its persistent drive connections. This feature makes transparent the complex commands needed to access network resources.

Following an initial sign-on to a specific network drive, File Manager remembers the navigation path implemented through potentially complex directory structures. This network drive connection is automatically generated in subsequent logons. Thus, applications can be developed that eliminate the need for users to remember convoluted access routines for different network drives.

A common concern of network users is performance. Windows can operate efficiently in a networked setting if it is configured properly. The implementor must allocate sufficient disk cache space on a network drive to accommodate execution of Windows programs. If programs must be read from regular hard disk at execution time due to inadequate cache space, then performance suffers.

Since Windows is heavily geared to a graphics environment, this may seem to represent another threat to peak performance. In actuality, however, Windows seeks access to a server only to initiate programs and save document files. Otherwise, all processing, including graphics, is centered on the individual PC.

There are omissions in recent Windows versions that may be important to some users. OLE capabilities that allow combining graphics, text, and data from different applications do not work in a network setting, although it is planned for a future release. In addition, peer-to-peer networking is not possible with Windows 3.1. It is more focused toward a server/client relationship. For peer-to-peer, Microsoft introduced Windows for Workgroups.

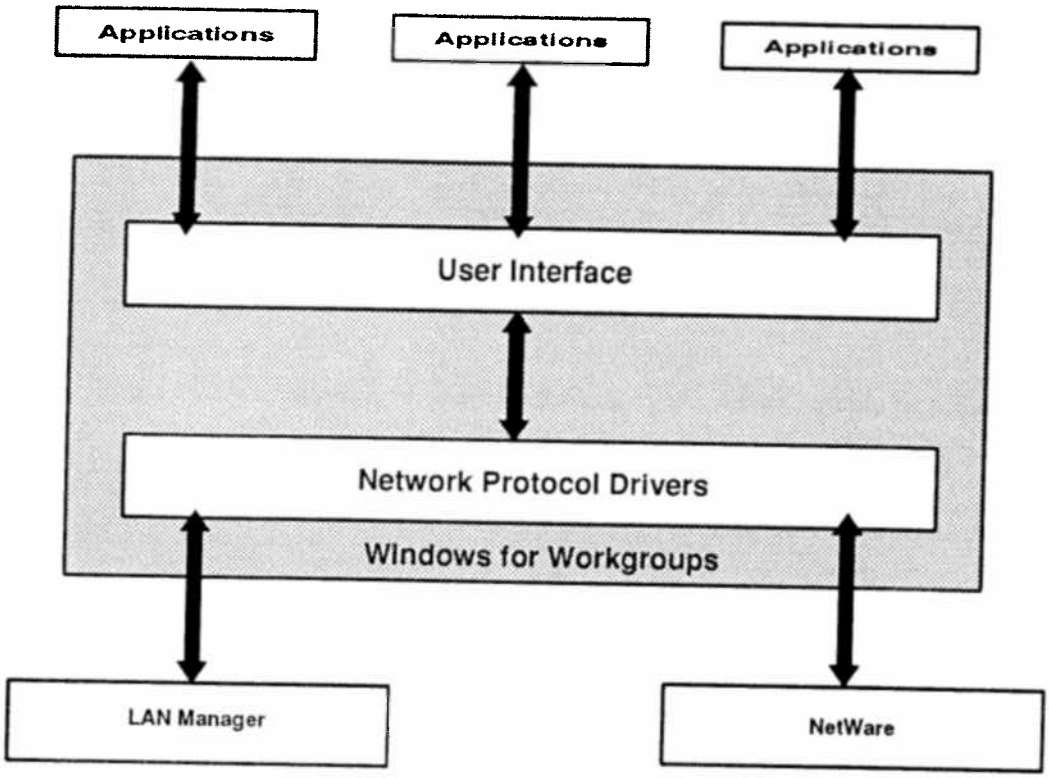
Microsoft's Windows for Workgroups Strategy

Windows for Workgroups is intended for small collections of PCs in small offices or for individual workgroups in larger organizations. In essence, it is Windows 3.x with peer-to-peer networking, E-mail, and group scheduling facilities added. These features serve to enhance communication between users on the network by providing a number of tools for exchanging messages, arranging group schedules and meetings, and providing file access across the network. Users of Windows 3.x will feel quite at home with the Windows for Workgroups user interface.

Windows for Workgroups uses a network protocol layer similar to the Windows printer and video drivers. Applications interact with the graphical user interface (GUI), which sends requests to the driver, which then calls specific network functions. Figure 2.2 shows the layout of this system.

The product is based on the concept of the workgroup, which may be defined as a group of professionals contributing different skills to the same project. These colleagues need access to many of the same project files, but are often contributing different information or analysis to these files. They will also have some data unique to their own areas of expertise, but will need to share this data with others, and to combine it with others' contributions into the completed project.

Figure 2.2 The Layout of Windows for Workgroups



The installation of Windows for Workgroups is very similar to that of Windows 3.x, except for some configuration options involving the networking adapter card and network interfaces. In many cases, Windows for Workgroups can also make intelligent guesses on the system and network configurations, and will only ask the installer to confirm the information. The operating system will attempt to connect to the network upon starting Windows.

Once users are in Windows for Workgroups, they will see little that is different from standard Windows 3.x. There is, however, a network addition to the Control Panel which lets users perform activities such as adding support for specific network adapter cards and adjusting the proportion of system time between local and remote users. There is also the ability to perform DDE between different workstations on the network.

Windows for Workgroups Bundled Applications

In addition to providing standard peer-to-peer networking services, Windows for Workgroups also provides several applications and utilities

that take advantage of the networked computing environment. The primary focus of these applications is to enhance communications among workgroup members. Other applications and utilities assist in user control over networking activities.

Probably the most important networking activity is electronic mail. E-mail is arguably the primary reason why most organizations decide to network computers in the first place. This makes E-mail the most important application to have available whenever a new networking standard appears on the market. Microsoft uses a version of its Microsoft Mail application as the communication centerpiece of Windows for Workgroups. Complementing Microsoft Mail is an implementation of the simple MAPI, which acts as a building block for messaging applications.

Windows for Workgroups also includes a group scheduling package, derived from Microsoft's Schedule Plus standalone product. This software enables groups on the same network to schedule meetings together and apply workgroup-wide deadlines, and it allows individuals to manage their own schedules online.

Other utilities include Chat, which enables two network users to hold a realtime, online conversation; Net Watcher, which tells a user who is accessing what files on the local system; and WinMeter, which displays the percentage of CPU time being allocated to remote users.

Networking Compatibility

One of Microsoft's goals with Windows for Workgroups was to ensure that it was compatible with existing network operating systems. There were two reasons for this. First, an important part of the company's networking vision involves integrating the workgroup into larger and more comprehensive networks. Microsoft would prefer that these larger networks use LAN Manager or Windows NT, but recognizes that by far most of the installed base of networks uses a non-Microsoft solution.

Second, many buyers of network operating systems already have networks installed at their sites, and compatibility is a primary buying

criterion. Sophisticated network buyers do not want to depend on a single vendor for their networking needs, but do need diverse solutions to work well together.

Appendix C covers Windows for Workgroups in greater detail.

Microsoft, therefore, is relying on its own resources and its clout in the market in order to provide connectivity with other network operating systems. The initial release of Windows for Workgroups included Novell NetWare drivers, and at least one other major vendor (Banyan Systems) has announced that it will include support for the product.

Windows NT and Networking

Windows NT takes networking several steps further than 3.1. Among its new features are:

- Versatility – NT can function well on workstations, desktops, network servers, and in client/server environments.

- Symmetric multiprocessor support is available when the host system contains multiple processors. NT will allocate tasks to the available processors.

- Preemptive multitasking is supported. This allows many programs to run concurrently, eliminating the need for Terminate-and-Stay-Resident (TSR) routines.

- Security in the form of passwords, and specific access permissions are incorporated, although early NT test versions have reportedly not been foolproof in this area.

- Virtual memory allows running larger programs than physical memory can normally support. Only the portion of a program actually being executed needs to be memory resident.

- A Simple Network Management Protocol (SNMP) package allows management of NT platforms from standard network management applications.

One of the realities that has become apparent to early NT users is its voracious (by today's standards) appetite for computing resources.

Microsoft proclaims that 8 MB of random access memory (RAM) and 100 MB of hard disk are sufficient for NT operation. Implementors find this scenario degrades performance dramatically. Realistically, an average user needs 16 MB of RAM and 150 to 200 MB of hard disk to support ongoing operations.

Windows NT as an Operating System

The previous section looked at NT from a networking perspective. The operating system itself offers several positive features, aside from the large compute resource needed for daily operations. In the ongoing "32-bit wars" NT will face its major competition from IBM's OS/2, Sun Microsystems' Solaris, and Unix System V, Release 4. The acquisition of Unix System Laboratories by Novell will propel Unix to increased success in the marketplace due to Novell's greater marketing strength. Microsoft should continue to dominate on the desktop, but at the server end all of the aforementioned operating systems will be strong contenders.

Built with a C language kernel, NT can be ported to diverse hardware platforms with varying degrees of ease. It will function primarily as a server on Intel and Reduced Instruction Set Computer (RISC) processors. NT is also scheduled to support DEC's Alpha RISC platform, and Hewlett-Packard's RISC machines and Sun's SPARCstations. OS/2 applications will also run under NT.

Some early NT testers have found problems in porting Windows 3.x applications to the new environment. Some of the migrated programs would not run under NT. Others demonstrated poor performance characteristics versus their execution on 3.x platforms.

Other early users have lamented the complexity of installing NT, particularly in a networked setting. Automatic network interface device detection routines are absent. The implementor, therefore, must deal with a series of input procedures asking questions about relevant I/O ports, shared memory addresses, and direct memory access channels. Unlike Windows for Workgroups, NT demands an experienced implementation team.

Windows and WOSA

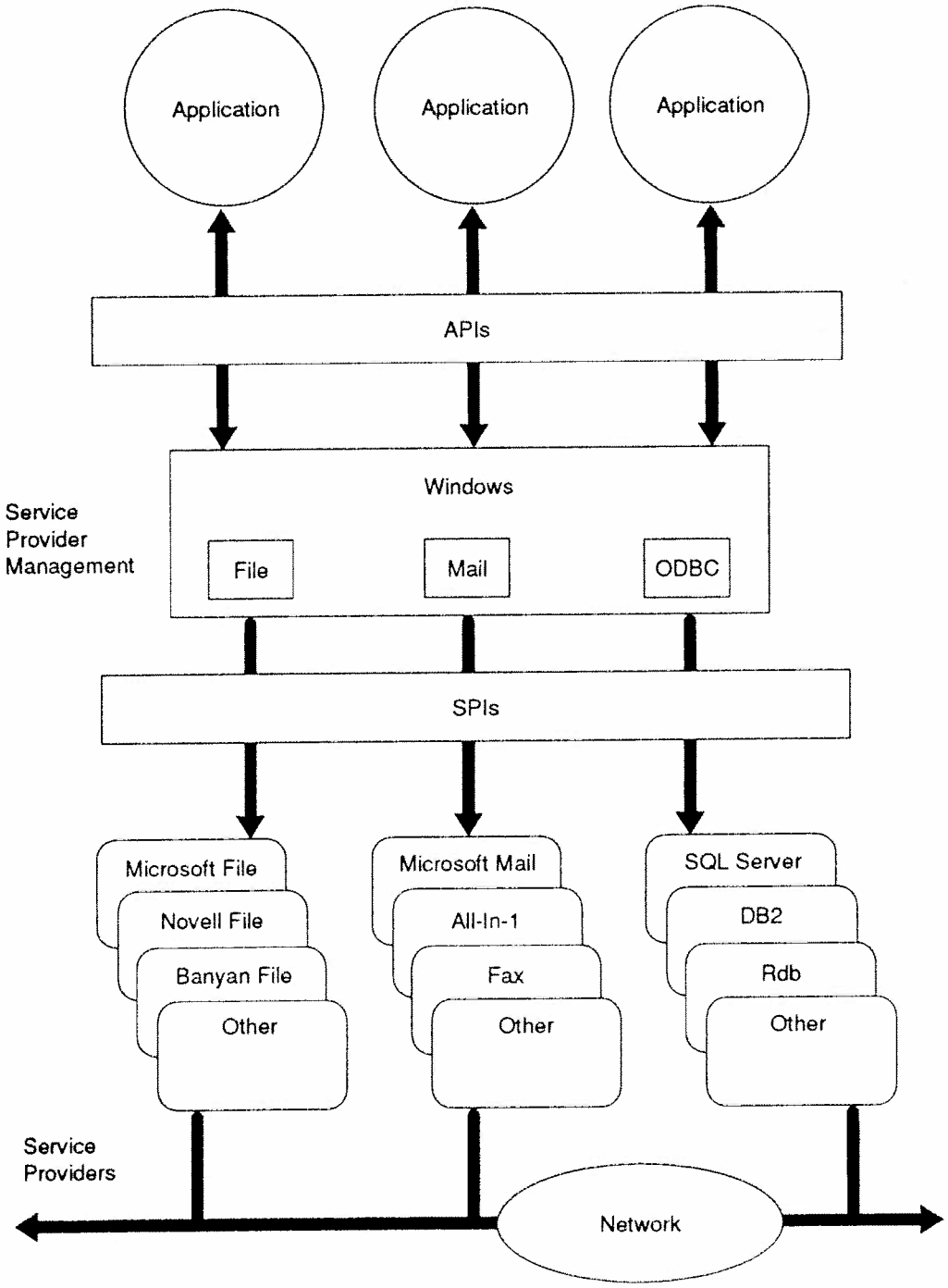
Windows presents users with a consistent application interface. Once users are comfortable with one application interface, others can be mastered quickly. Similarly, WOSA provides developers of distributed applications with a single interface for interacting with service providers such as database managers and E-mail modules. WOSA application programmers need learn only a single set of APIs for all implementations of a particular service, e.g. database managers (see Figure 2.3).

In addition, WOSA supports 3.x and NT versions of Windows. Both implementations of the operating system can access information and services across heterogeneous computing environments — all from one consistent Windows graphical user interface. WOSA supports interoperability through a collection of open interfaces for key system-wide services developed by Microsoft and other service providers. Accordingly, as part of WOSA, the networking architecture of Windows has been modified to make it more extensible. Standard, open interfaces have been defined for network services such as electronic mail and database access, and more will be specified by Microsoft and other developers for file sharing, printer sharing, administration, and configuration.

Future Directions

The future of Windows, of course, is inexorably linked to the ongoing development of NT. Early in its existence, NT became the de facto standard for supporting applications operating on a variety of RISC and Intel chips.

Figure 2.3 WOSA Single Set APIs



Windows NT will be one of the more "open" proprietary systems in the marketplace, although IBM's OS/2 and DEC's VMS have also been moving in this direction. Unix's reputation for openness is already well known, albeit a bit overstated. With WOSA, however, and APIs such as Windows Sockets to connect to TCP/IP, POSIX, and external LANs, NT is well along in its quest for widespread platform interoperability.

Even though some estimates place NT's market potential in excess of two million nodes, this is merely speculation. The factors that will determine its success are pricing and reliability of early NT versions. Microsoft's legendary marketing prowess will only carry NT so far. User operating experience will determine the rest.

Chapter 3

WOSA Architecture

As a software architecture, WOSA is a relatively straightforward technology model. Figure 3.1 depicts the major elements of this model where user applications invoke specified APIs as appropriate to the functional service being sought, e.g. messaging service. The actual service provider, in this case MAPI, is accessed through SPIs developed for specific messaging functions. If the functional service is at some point replaced or modified, then SPIs will be altered accordingly. User applications, however, remain stable and may be unaware of changes implemented at the functional service end.

On a more detailed level, there are software drivers appended to the Windows operating system that aid the WOSA process (see Figure 3.2). Labeled WOSA drivers, they are used by SPIs to access a particular type of back-end functionality. These Windows system extensions support operations such as messaging, addressing, and data transport.

Current Capabilities

WOSA is extensible both in the types of functional services it can provide and in the quantity of any given service it can support. Any list of current capabilities, therefore, is transitory and subject to change on a regular basis. WOSA currently supports the following APIs:

- Open Database Connectivity
- Messaging
- License Service
- Windows Sockets
- Windows SNA
- Remote Procedure Call
- Extensions for Financial Services

New functional areas for the future include distributed file and security systems, systems management, data access, communications, and directory services.

Figure 3.1 Major Elements of WOSA

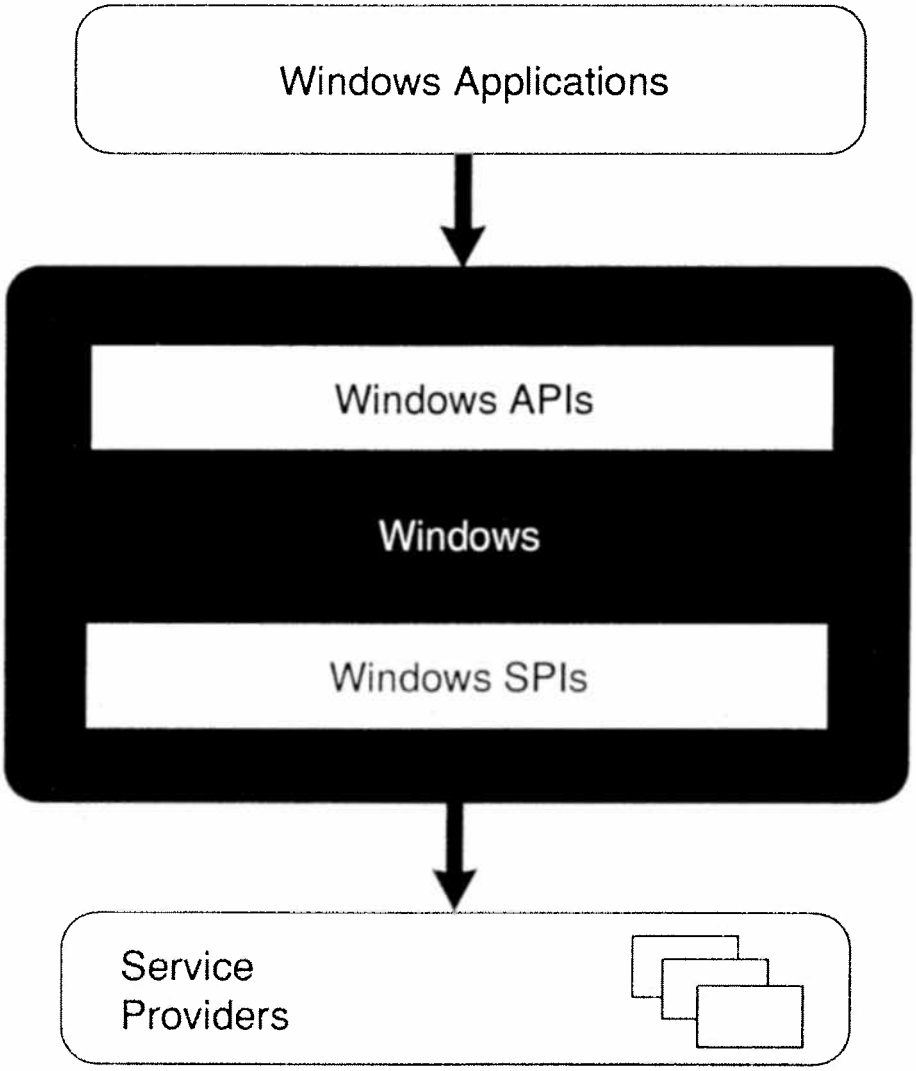
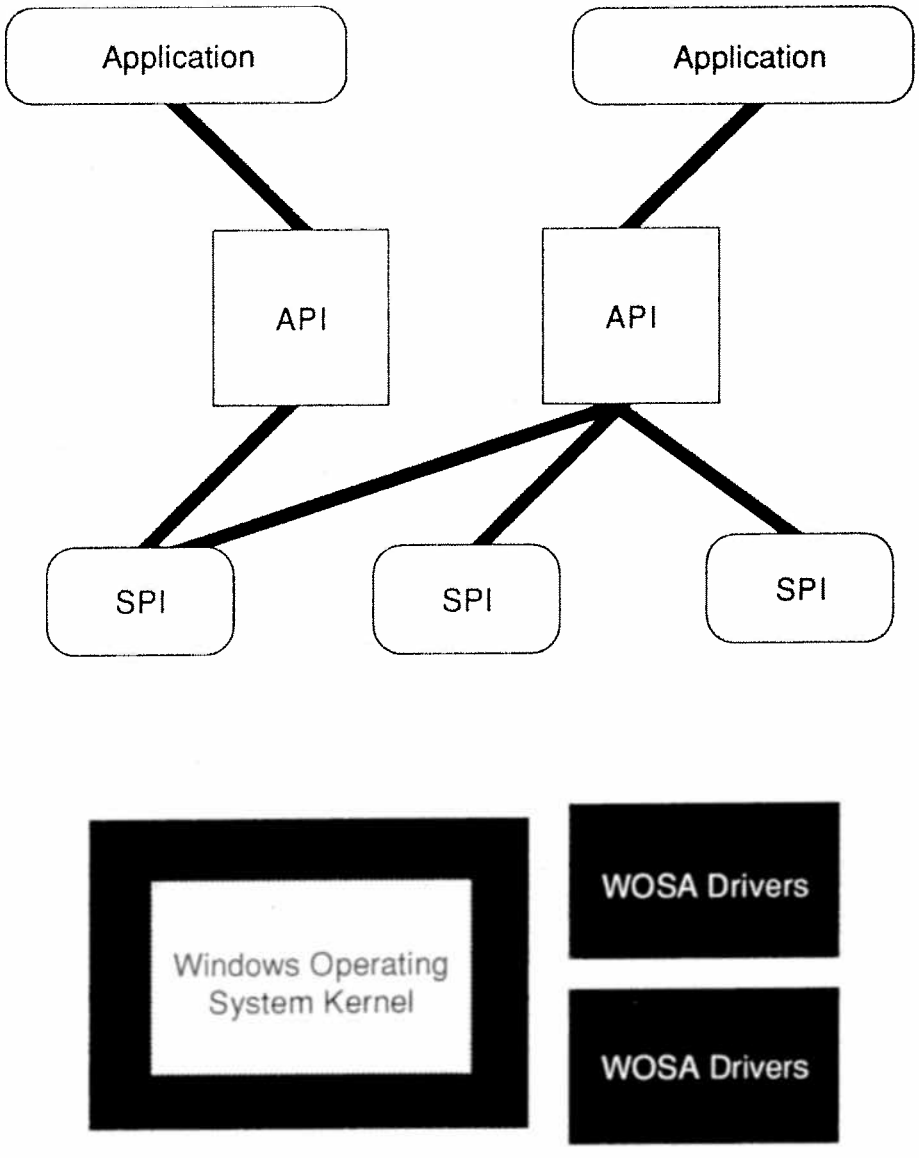


Figure 3.2 WOSA Process



Distributed Systems

One of the most prominent vehicles in support of distributed systems is the Distributed Computing Environment from OSF. The DCE presents a set of tightly-integrated technologies to enable vendors and users to provide transparent computing in heterogeneous environments. It also includes services necessary for the development and maintenance of distributed applications.

Distributed computing is a complex subject. In order to develop an efficient distributed configuration, a plan for communication, interaction, and resource control is required. OSF's DCE provides a model and related tools to help the user build distributed applications.

The various elements of DCE as portrayed in Figure 3.3 are:

- RPC and Presentation Services – Interface definition languages and RPCs enable programmers to transfer control and data across a network in a transparent manner. This helps to mask the network's complexity.

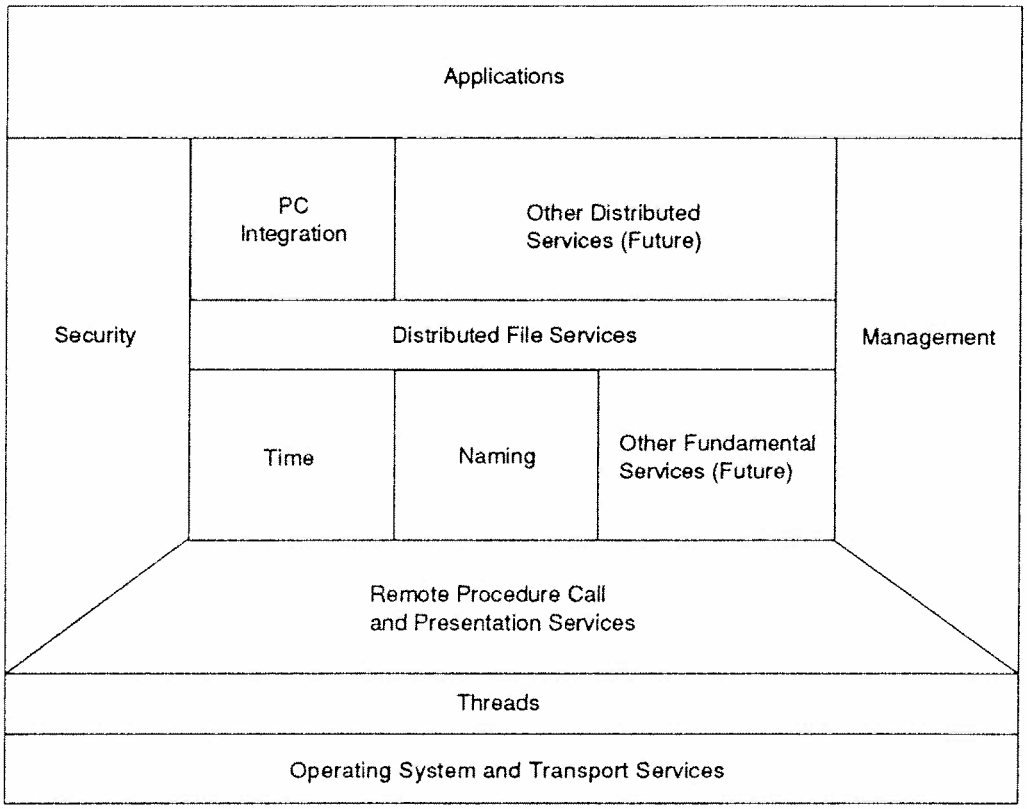
- Naming – User-oriented names specifying computers, files, and people must be easily accessible in a distributed environment. Referred to as a directory service, it must be standard in appearance and rules for all clients.

- Security – Distributed applications and services must be able to identify users, control access to resources, and guard the integrity of all applications.

- Threads – Represents a method of supporting parallel execution by managing multiple threads of control within a process operating in a distributed environment.

- Time Service – Synchronizes the clocks of all systems in a distributed environment so that executing applications can operate correctly.

Figure 3.3 Distributed Computing Environment Architecture



- Distributed File System – This component extends the local file system to the network in order to allow users full access to files on remote configurations.

- Personal Computer Integration – Allows PCs using MS-DOS to access file and print services outside the DOS environment.

- Management – Partially addressed in the preceding elements, its sheer complexity in a distributed, heterogeneous configuration has led to an entirely new project entitled OSF Distributed Management Environment.

Table 3.1 lists the specific products that comprise OSF's DCE. Also shown are the products associated with another prominent distributed model: Sun Microsystems Open Network Computing (ONC+).

Table 3.1 Products of OSF's DCE

DCE Element	OSF/DCE	SUN ONC+
RPC	DEC/H-P NCS, NDR	Sun RPC, XDR
Naming	Digital DECdns Siemens DIR-x	Sun NIS
Security	MIT Kerberos	MIT Kerberos Sun Secure RPC
Threads	DEC CMA	Sun RPC
Time Service	Digital DECdts	NTP
Distr. File System	Transarc AFS	Sun NFS
PC Integration	Sun PC-NFS LAN Manager/X	Sun PC-NFS

NCS– Network Computing System
CMA– Concert Multithread Architecture
AFS– Andrew File System
XDR– External Data Representation
NDR– Network Data Representation
NTP– Network Time Protocol

DCE defines a standardized service environment that is to be implemented individually on each operating system platform. Organizations planning to migrate to this capability will need a mechanism to support the move from one platform to another. WOSA complements DCE by integrating various distributed systems in a transparent manner. The end-user continues to view a consistent interface, no matter the underlying distributed scheme and platform.

Multiple Service APIs

It is important to understand the distinction between a set of APIs providing multiple services, as manifested by WOSA, and a group of APIs supporting multiple platforms, but focused on one service offering. An example of the latter would be VIM, the messaging module for cross-platform service.

VIM and products like it do indeed solve the problem of cross-platform development. They fail, however, to deal with the larger issue of multiple system services. Among these service requirements are memory

management, graphics, multitasking and multithreading, print services, etc. Offerings such as VIM address only one aspect of system operations.

The goal of WOSA is to deal with the complete range of APIs necessary to support a full-featured system environment. It will be several years before WOSA approaches such a lofty goal, but its vision far exceeds the one-issue solutions presented by packages in the VIM category.

Universal Client

The principles supported by WOSA have been emulated by various vehicles in the past, including the Streams environment. WOSA, however, offers a wider variety of services than available from these earlier efforts. In addition, WOSA's architecture is focused on a higher layer in the logic chain. These older vehicles operate at the lower layers, whereas WOSA functions at a higher service layer.

Due in part to WOSA's breadth of service offerings, the Windows operating system has approached the status of universal client in the overall scheme of network architectures. WOSA also enforces a consistent programming interface via APIs, not allowing gradations or levels of access to operating system services. Thus, with Windows' ubiquity and WOSA's structured conventions, the status of universal client, at least in the Windows environment, has been achieved.

Chapter 4

Open Database Connectivity

The ODBC interface allows applications to access information in DBMS modules via Structured Query Language (SQL) commands. The latter is a standard database access methodology accepted worldwide.

With ODBC, the Windows application developer achieves a high degree of interoperability. No specific DBMS must be targeted in the compiled program. Rather, each application can interact with any SQL DBMS, depending on the needs of a particular installation. Each user adds database driver routines that service the specific DBMS packages operational within that installation. The application program itself remains unchanged regardless of the DBMS being used.

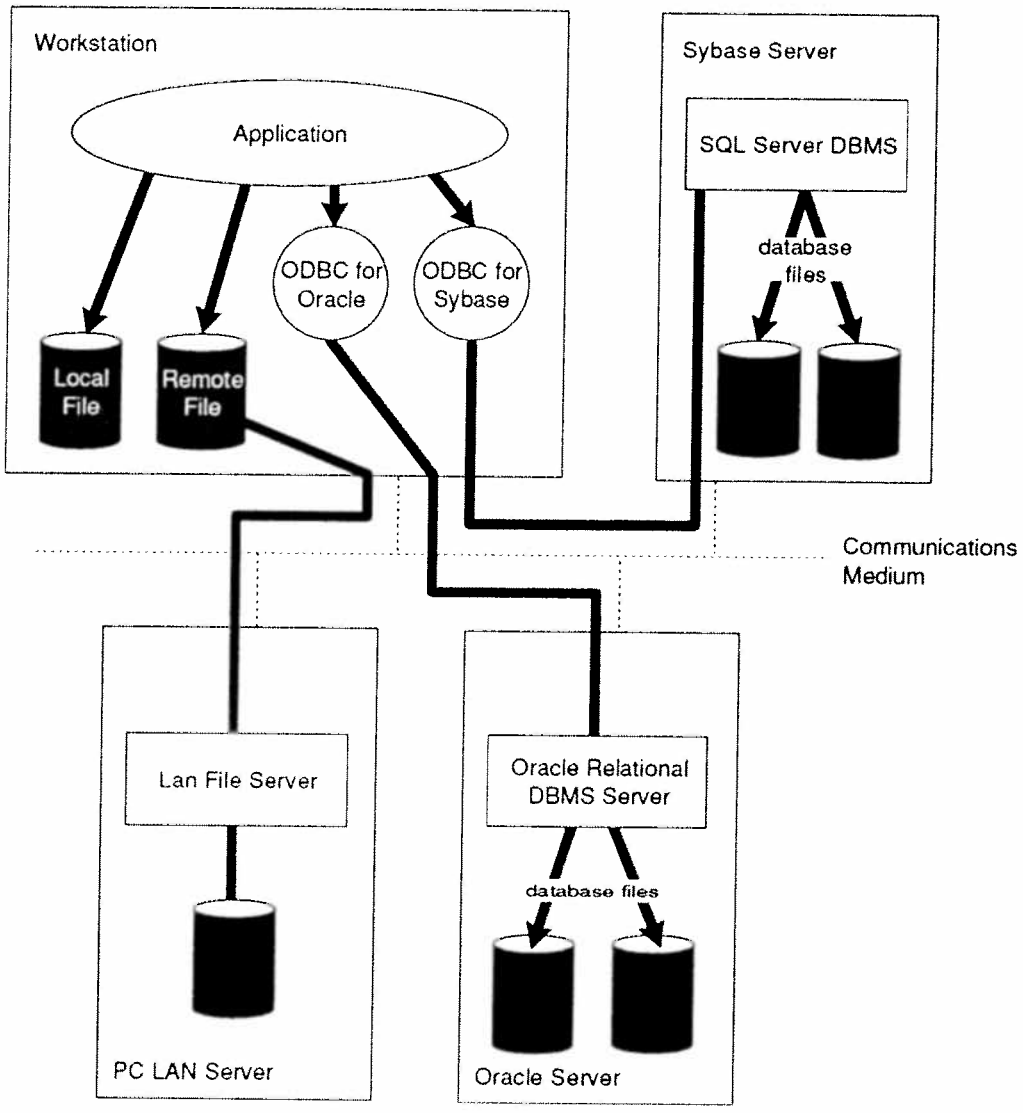
History

ODBC provides a standard interface that enables applications and data repositories to exchange information in a dynamic manner. Since there are numerous data protocols, communications schemes, and DBMS capabilities now and for the foreseeable future, an interface methodology was needed that could adapt to changing circumstances at runtime. The ODBC solution is to invoke DLLs that can respond to a particular Data Source via a specific communication method on demand during program execution.

Prior to the arrival of vehicles such as ODBC, applications typically performed database tasks in conjunction with a specific database, e.g. IBM's DB2, Oracle, etc. Such applications most often employed embedded SQL statements to achieve their mission. This is otherwise efficient, but forced developers to rewrite their applications if the database being accessed changed.

In a large organization, an application may have to access more than one database type in order to complete its task (see Figure 4.1). Under a non-ODBC approach, multiple versions of the accessing application would be needed to access each database. The ODBC interface accommodates unique databases via DLLs at runtime.

Figure 4.1 The ODBC Interface



ODBC Basics

ODBC is not an actual software product. It is a software interface specification. Users of ODBC, i.e., DBMS providers and application developers, follow the modality of the ODBC specification. Applications invoke the ODBC API as a client, whereas DBMS packages use it as a service provider.

Several DBMS vendors provide client/servers DLL for Windows, though, unlike a standard ODBC DLL, their interfaces to the client on a PC are vendor-specific. The user cannot change the database without changing the application because the interfaces are non-standard. This differs from the ODBC approach already described. A standard ODBC interface defines the following:

- SQL syntax based on the SQL Access Group (SAG) and X/Open's SQL specification of 1991;
- a library of ODBC function calls that allow an application to connect to a DBMS, execute SQL commands, and retrieve results;
- a set of error codes;
- a standard representation for multiple data types;
- a standard procedure for connecting to and logging on to a DBMS.

The ODBC interface offers several flexible features. For one, the same object code can access different DBMS packages. It also supports the creation of SQL statements on the fly at runtime. Additionally, ODBC allows data values to be sent and retrieved in a format convenient to an application.

Two types of function calls are associated with the ODBC interface. The first features basic capabilities derived from X/Open and SAG's CLI specification. The second function call type includes proprietary extensions dealing with issues such as asynchronous processing and scrollable cursors. Using the latter function call type diminishes an application's interoperability capabilities.

The ODBC architecture consists of four major components. As illustrated in Figure 4.2, there is a functional application that initiates the process by calling ODBC services which transmit SQL commands and subsequently receive requested information.

There are various Drivers which are activated by the Driver Manager on behalf of an application. Each Driver processes ODBC function calls, forwards SQL requests to the appropriate Data Source, and returns results to the initiating application. A Driver performs whatever syntax modification to the request that may be needed in order to conform to the target DBMS.

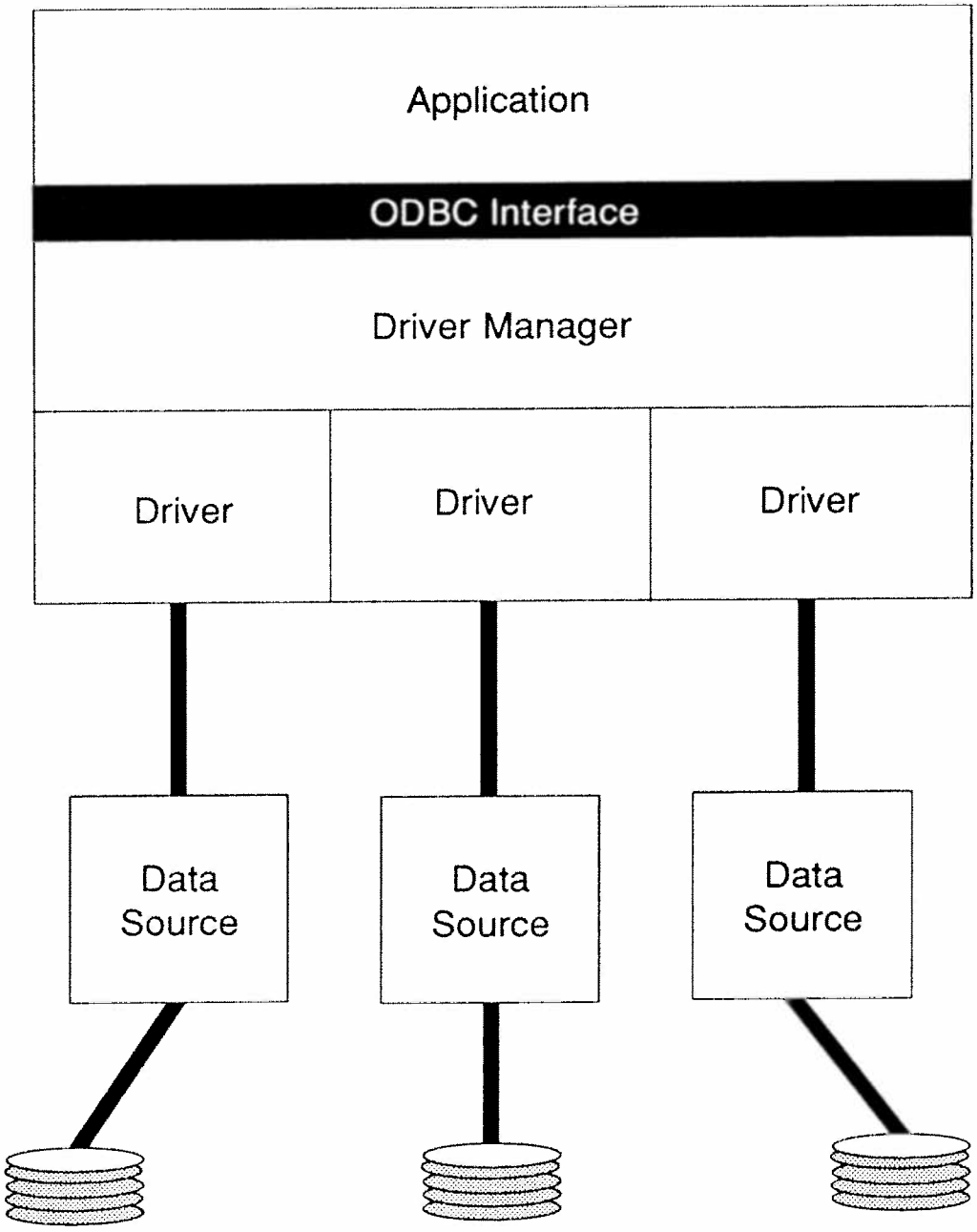
The Data Source depicted in Figure 4.2 consists not only of the DBMS being accessed, but also its operating system and support environment. If it is on a network remote from the initiating platform, its associated network platform is also part of this scenario.

From the viewpoint of an application, the distinction between a Driver and Driver Manager is non-existent. They simply appear as an entity that processes ODBC function calls.

In more precise terms, an application using the ODBC interface performs the following tasks:

- requests a connection to a Data Source
- forwards SQL requests to that Data Source
- sets up storage areas and data formats for the response to SQL requests
- requests results
- handles errors as they occur
- reports results back to a user if appropriate
- requests commit or rollback operations for transaction control
- terminates the Data Source connection

Figure 4.2 ODBC Architecture



The Driver Manager, as described earlier, has as its primary mission the loading of Drivers. It is also a DLL with an import library that does the following:

- uses the ODBC.INI file to map a specific Data Source to a particular Driver DLL;
- manages multiple ODBC function calls;
- provides parameter validation and sequence validation for ODBC calls;
- loads a Driver when an application initiates the SQLCONNECT or SQLDRIVERCONNECT commands.

Drivers are DLLs that implement ODBC function calls and interact with a Data Source. The latter could be local or remote, utilizing one of several operating systems and network configurations, and feature one of the data storage products listed in Table 4.1.

Drivers also perform the following specific tasks:

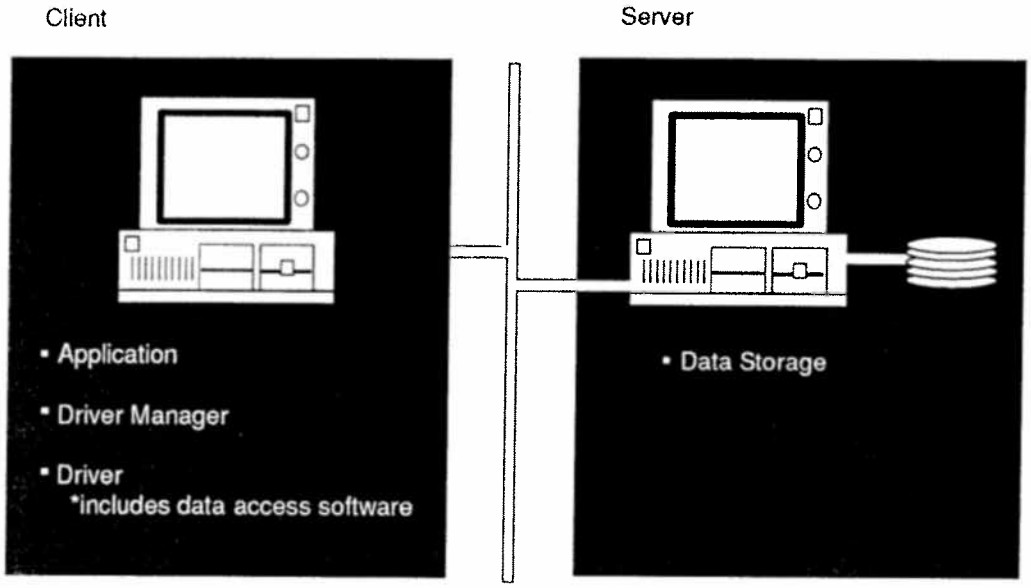
- establish a connection to a Data Source;
- submit requests and retrieve results from those requests for transmission to an application;
- perform data format conversions as required;
- report error status information to the application.

There are two types of Drivers. A single-tier Driver processes both ODBC function calls and SQL statements. It accesses the target database directly. There is no intervention by servers or similar third-party agents. Figure 4.3 illustrates the single-tier approach.

Table 4.1 Partial list of ODBC Drivers Currently under Third-Party Development

DBMS	Company
DAL Servers	Apple
IBM AS/400	Born Software Development Group
Model 204	CCA
Cincom	Cincom
Integra-SQL	Coromandel
DP4	Datafit
DEC Rdb	DEC
Siemens/Nixdorf	GFS & Siemens
Allbase	Hewlett-Packard Company
SQLBase	Gupta Technologies, Inc.
EDA/SQL	Information Builders, Inc.
Informix	Informix Software, Inc.
Ingres	Ingres
multiple	IQ Software
mdbs IV	mdbs, Inc.
IBM DB2 (w/Microsoft)	MicroDecisionware, Inc.
R:base	Microrim, Inc.
NOMADGateway	Must Software
Sharebase	NCR/Sharebase
Pick DBMS	Paradigm
Porting all Q+E drivers	Pioneer Software
Progress & Progress SQL	Progress Software Corporation
Quadbase/SQL	Quadbase
Revelation	Revelation Technologies
IBM AS/400	Rochester Software Connection, Inc.
ENTIRE Natural SQL Server	Software AG
ENTIRE SQL-DB	Software AG
ENTIRE ADABAS SQL Server	Software AG
Sybase SQL Server	Sybase, Inc.
Tandem NonStop	Tandem Computers, Inc.
Techgnosis Servers	Techgnosis, Inc.
RMS	Vertisoft Research, Inc.
Watcom SQL	Watcom

Figure 4.3 Single-Tier Driver

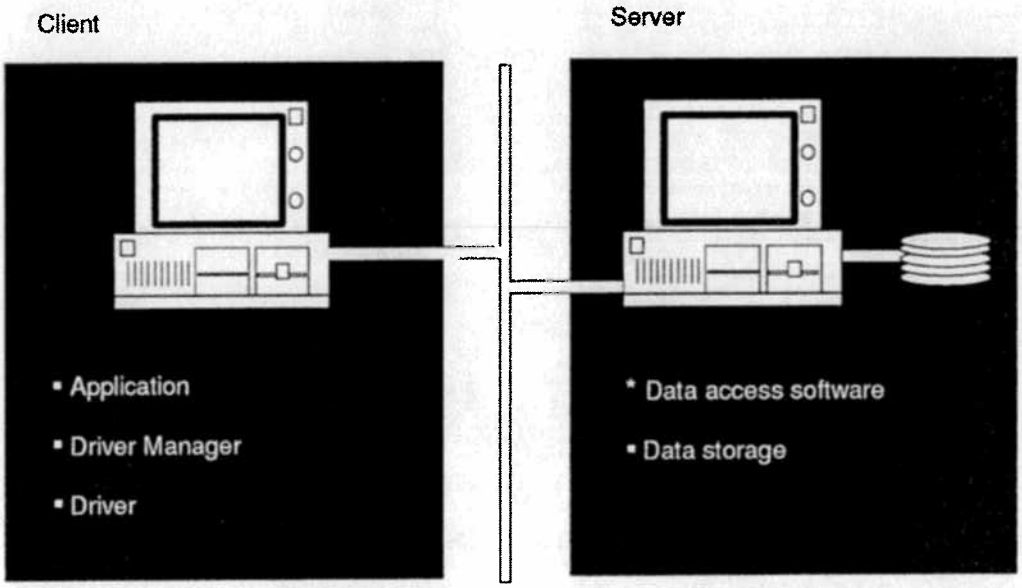


In a multiple-tier Driver configuration, SQL requests are forwarded to a server for further processing. The actual data access function emanates from a server, not from the Driver itself. The entire system may reside on a single unit, or may be dispersed across platforms as depicted in Figure 4.4.

Thus, there is flexibility in the ODBC interface. Interoperability is also enhanced since an application need not target a specific Data Source at the time of program development. Users can add Drivers to an application after it is compiled and has achieved initial operational readiness.

The ODBC API defines a set of core and higher functions that match X/Open and SAG CLI specifications. These levels of conformance are listed in Table 4.2.

Figure 4.4 Multiple-Tier Driver



Similarly, ODBC defines a set of grammar levels that correspond with X/Open and SAG SQL specifications of 1991. SQL conformance level information is shown in Table 4.3.

In summary, ODBC has four major functions:

1) It implements SQL statements for data access.
2) It provides concurrent access to multiple databases.
3) It maintains listing of available databases.
4) It encompasses a DLL-based API that each ODBC Data Source must support.

Close linkage to SAG's SQL document places ODBC in the mainstream as far as standards are concerned. This specification is not "reinventing the wheel." Rather, it is implementing that "wheel" in a functional role.

Table 4.2 ODBC API Conformance Levels

Core API	Allocate and free environment, connection, and statement handles.
	Connect to Data Sources. Use multiple statements on a connection.
	Prepare and execute SQL statements. Execute SQL statements immediately.
	Assign storage for parameters in an SQL statement and result columns.
	Retrieve data from a result set. Retrieve information about a result set.
	Commit or roll back transactions.
	Retrieve error information.
Level 1 API	Core API functionality.
	Connect to Data Sources with driver-specific dialog boxes.
	Set and inquire values of statement and connection options.
	Send part or all of a parameter value (useful for long data).
	Retrieve part or all of a result column value (useful for long data).
	Retrieve catalog information (columns, special columns, statistics, and tables.
	Retrieve information about Driver and Data Source capabilities, such as supported data types, scalar functions, and ODBC functions.
Level 2 API	Core and Level 1 API functionality.
	Browse available connections and list available Data Sources
	Send arrays of parameter values. Retrieve arrays of result column values.
	Retrieve the number of parameters and describe individual parameters.
	Retrieve the native form of an SQL statement.
	Retrieve catalog information (privileges, keys, and procedures).
	Call a translation DLL.

Table 4.3 SQL Conformance Levels

Minimum SQL Grammar	Data Definition Language (DDL): CREATE TABLE and DROP TABLE.
	Data Manipulation Language (DML):simple SELECT, INSERT, UPDATE SEARCHED, and DELETE SEARCHED.
	Expressions:simple (such as A>B+C).
	Data types:CHAR.
Core SQL Grammar	Minimum SQL grammar.
	DDL:ALTER TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, GRANT, and REVOKE.
	DML:full SELECT, positioned UPDATE, and positioned DELETE.
	Expressions:subquery, set functions such as SUM and MIN.
	Data types:VARCHAR, DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, FLOAT, DOUBLE PRECISION.
Extended SQL Grammar	Minimum and Core SQL grammar.
	DML:outer joins.
	Expressions:scalar functions such as SUBSTRING and ABS, date, time, and timestamp literals.
	Data types:LONG VARCHAR, BIT, TINYINT, BIGINT, BINARY, VARBINARY, LONG VARBINARY, DATE, TIME, TIMESTAMP.
	Batch SQL statements.
	Procedure calls.

Applications can now access different databases as needed for various tasks — as long as the target database conforms to ODBC conventions. A user may access Oracle, then Sybase, then Informix databases on various occasions as required. As long as the database is ODBC-compliant, the application itself remains unchanged.

Another important ODBC feature is its ability to concurrently interact with multiple databases. This allows ODBC clients to co-mingle and collate data from multiple sources, plus update several remote Data

Sources at the same time. In effect, this allows ODBC to manage the problems of data dispersal, to control a distributed environment.

Industry support for ODBC has been impressive. Table 4.4 identifies early backers of the specification. New adherents are being added at a steady pace, and de facto standard status is close to being achieved.

Table 4.4 Early Supporters of the ODBC Specification

- Apple Computer, Inc.
- Bull HN Information Systems
- Cincom Systems, Inc.
- Computer Corp. of America
- Coromandel Industries, Inc.
- DEC
- EASEL Corporation
- Fox Software, Inc.
- Fucrum Technolgies, Inc.
- Gupta Technologies, Inc.
- Hewlett-Packard
- Information Builders, Inc.
- Informix Software, Inc.
- Ingres
- IQ Software
- Lotus Development Corporation
- mdbs
- Micro Decisionware, Inc.
- Microsoft Corporation
- Microrim, Inc.
- NCR/Teradata
- Neon Systems, Inc.
- Novell, Inc.
- Oracle Corp.
- PageAhead Software Corp.
- Pioneer Software
- Progress Software Corp.
- Raima Corporation
- Retix
- Rochester Software Corp.
- Sybase, Inc.
- Tandem Computers, Inc.
- Uniface Corporation
- Unify Corporation
- Watcom

SQL Standard

Early database applications were run on large mainframes. User access was provided by "dumb" terminals, whose actions consisted mainly of sending batch requests for subsequent retrieval of blocks of data.

The introduction of PCs with enormous processing power and storage changed all this. Software such as dBASE II and Lotus 1-2-3 enabled users to define and manipulate their own data, separate from the mainframe database. User "independence" had arrived.

The emergence of LANs and yet even more powerful PCs and workstations gave rise to the next phase of database computing. User devices were now linked together. File servers were introduced to support sharing of data among widespread platforms, with the user device doing most of the processing following data retrieval.

Eventually, file servers became bottlenecks as LANs increased their user population. File servers also did little in the areas of security, network

control, and data integrity. Thus, the next stage of database operations emerged: client/server computing, relational DBMS (RDBMS), and SQL.

SQL was originally defined by IBM for its SQL/DS and DB2 products. Other vendors have now adopted IBM's basic premise, but have incorporated extensions that inhibit interoperability. The need for standards became critical.

Both the American National Standards Institute (ANSI) and the International Standards Organization (ISO) have created working groups to define SQL standards. The resulting ANSI/ISO standard included an earlier ANSI X3.135–1986 document, along with Addendum 1. The latter outlines the syntax and semantics of Referential Integrity constraints.

In 1989, a new edition of the standard was released. It was called ANSI Database Language SQL with Integrity Enhancement. This specification detailed the syntax for Data Definition Language commands and Data Manipulation Language commands, plus embedded SQL commands within a host language.

The 1989 standard lists two levels of conformance to its mandates. Level 2 is the full SQL package. Level 1 is a subset of Level 2, and represents a cross-section of prevailing SQL implementations available then. SQL2 and SQL3 variants have either been released (SQL2), or remain in the planning stage (SQL3). ANSI and ISO continue to cooperate in producing these latest versions.

Meanwhile, the SQL Access Group strives to resolve differences among commercial implementations of the language. Some industry leaders do not participate in SAG, so its ultimate success is uncertain.

Other players in the SQL standards arena include X/Open. Its specifications roughly follow ANSI specifications, with particular attention given to Unix environments. ISO's Remote Database Access (RDA) standard also is designed to facilitate access to server databases from client devices.

When reviewing products associated with SQL, one must also take into account the databases to which they are linked. SQL does not exist in a vacuum. It functions in conjunction with an information repository. The SQL products in Table 4.5 are not all-inclusive, they are only representative of offerings available in the marketplace.

Table 4.5 Representative SQL Product Offerings

Product	Vendor
SQL Server	Sybase
SQL Server	Microsoft/Sybase
SQL Base	Gupta Technologies
Informix Online	Informix Software
NetWare SQL	Novell
Oracle Server, Oracle7	Oracle
XDB-Server	XDB Systems
Interbase	Borland/Interbase
Intelligent Database	Ask (Ingres Division)
Rdb	DEC
OS/2, APPC Server	IBM

SQL consists of three core components, the DDL, DML, and Data Control Language. The last is basic in its concept. It can *grant* or *revoke* access. There are refinements as to how these commands may be applied. For example, user groupings may be specified in some implementations.

DDL can create tables, indexes, and views, the latter a logical transparency defined for a specific need during data access. Tables contain real data and indexes aid information retrieval. Indexes can assist with range searches, so-called wild card searches, ordering, along with direct retrievals and updates.

Most users will invoke SQL in one of two ways. Static SQL is found in stable application programs that are used repeatedly without change. Except for parameters, SQL statements contained within these programs do not vary from one execution to another.

Dynamic SQL is generally transparent to the end-user, i.e., it is found in system software that must handle a changing environment. For example, database browsing is managed by interactive query support software that meets changing user demands. Occasionally, even application software might need the facilities of dynamic SQL, but generally system software handles all the situations which compel its use. Commands such as SQLDA (SQL dynamic area) and DESCRIBE (fills in SQLDA) deal with dynamic SQL situations.

Several trends are evident for SQL, along with the databases to which it is linked. Most of the new features are part of SQL2 and SQL3 standards, the latter not yet an official standard, but still in the development stage. New features include:

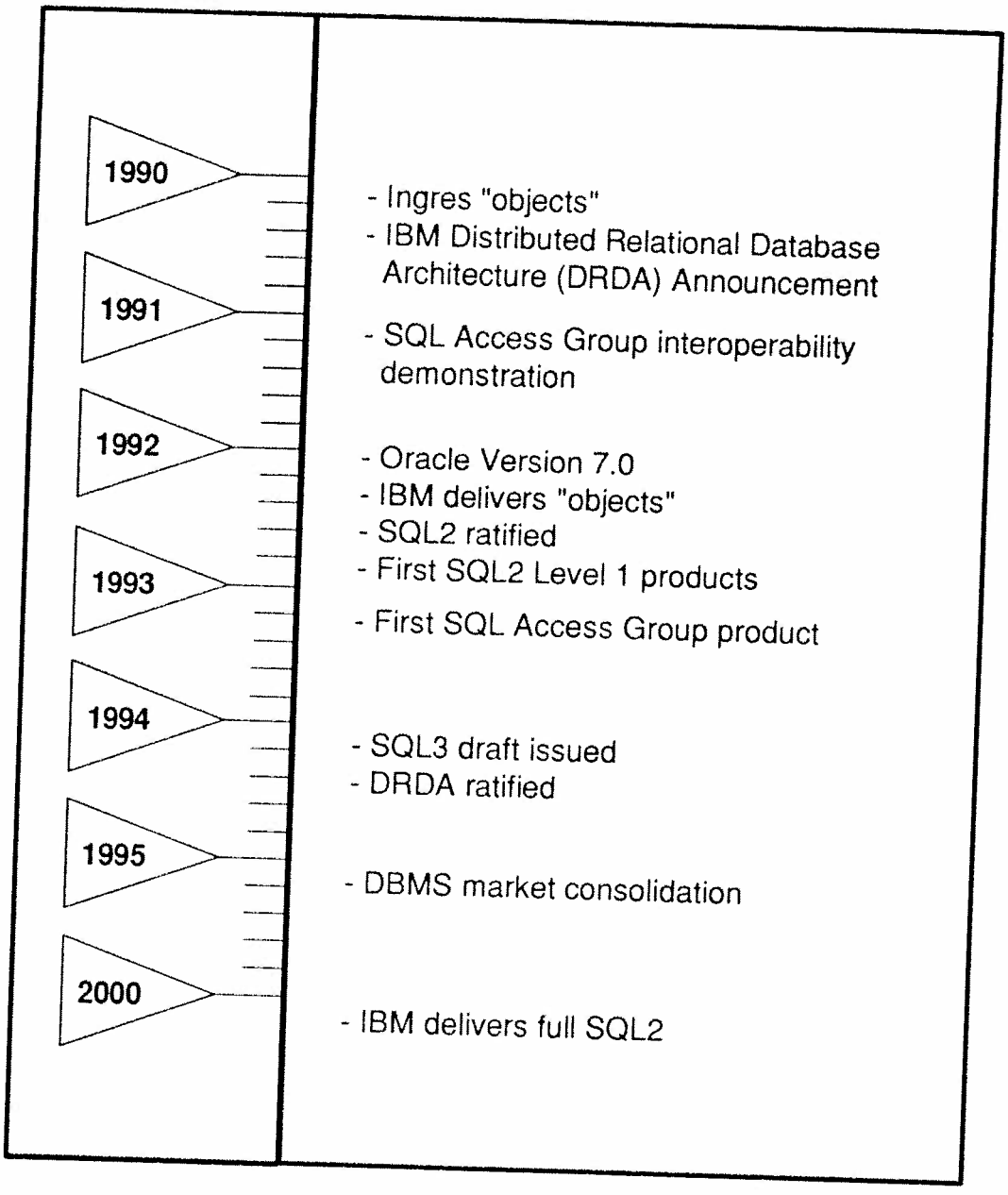
- Adding control structures to SQL to permit it to deal with larger blocks of application logic.

- Adopting object-oriented features into the SQL language that are now found in object-oriented databases.

- Integrating inference rules into SQL in order to support features like recursion.

- Adding multimedia data types and their access mechanisms into SQL. Several vendors have inserted some form of one or more of these new features into their products. As SQL3 inches closer to official acceptance as a standard, yet further commercial offerings will incorporate SQL extensions, particularly in the area of object-oriented technology. Figure 4.5 illustrates a projection of SQL developments during this decade.

Figure 4.5 SQL Development Time Line



Source: Gartner Group

SQL is a volatile technology. The search for standards manifested in the work of ANSI, ISO, SAG, X/Open, and others continues unabated. What began in the 1970s as an attempt to facilitate database access has evolved in the 1990s as a group of differentiated products.

While the core functionality of SQL offerings are generally harmonious, "extracurricular" extensions present a major problem. Most databases still exist as islands of information, requiring different access procedures to retrieve and manipulate data. ODBC is yet one more effort to harmonize data access problems for the user. It appears to have an excellent chance of accomplishing that goal.

Application Development

An application initiates the following steps when interacting with a Data Source:

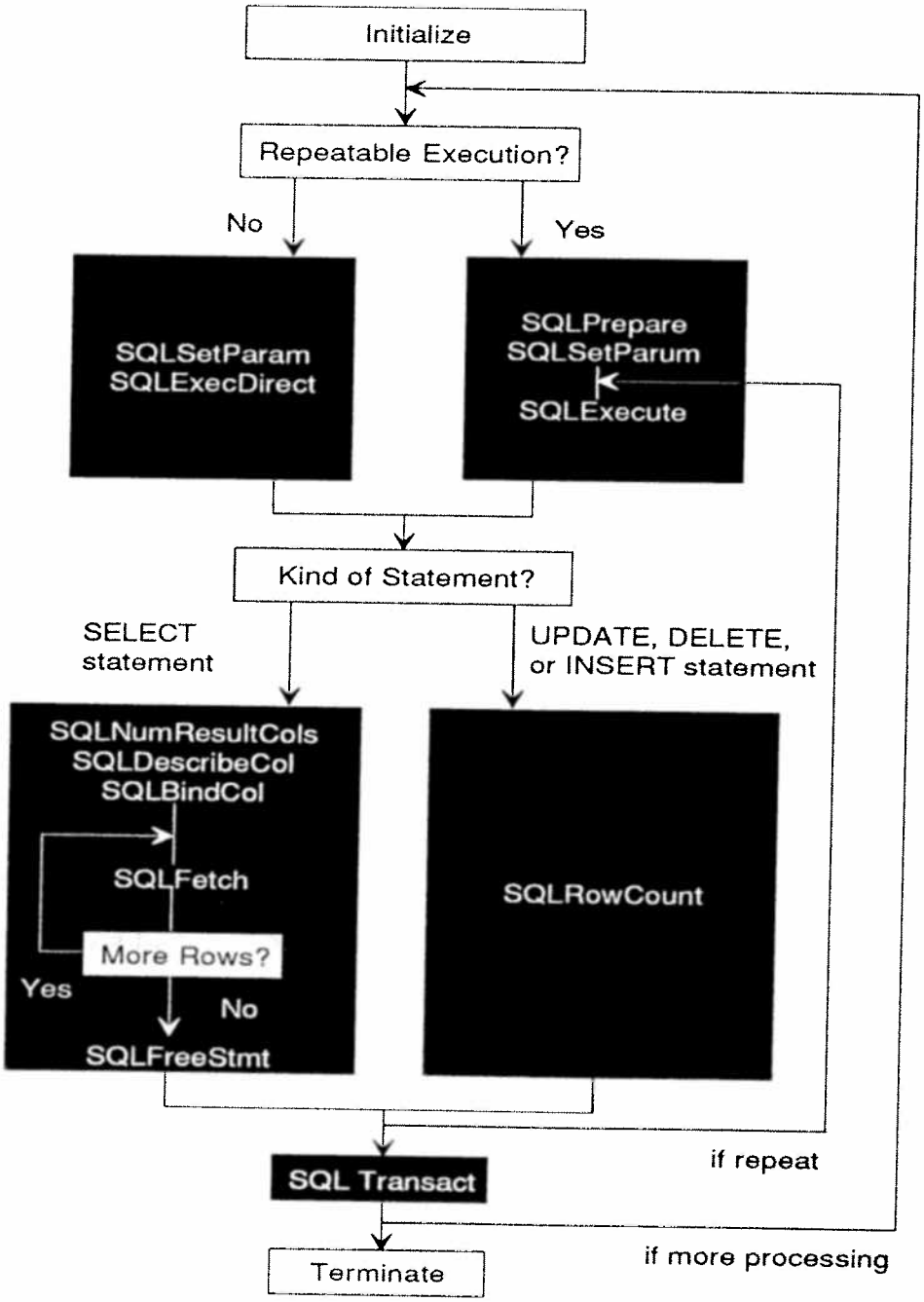
- Connects to the Data Source by specifying its name and associated identifiers.
- Processes SQL statements. Includes moving the SQL text string to a buffer, submitting the statement for prepared (will use same SQL statement later) or immediate execution, handling error information received from the Driver and implementing appropriate action.
- Completes each transaction by either committing it or rolling it back, depending on the success of the overall process.
- Terminates the Data Source connection upon completion of assigned task.

Figure 4.6 depicts a basic sequence of ODBC function calls as they execute SQL statements, the very core of application processing.

Connecting to a Data Source

A Data Source consists of the data a user wishes to retrieve, its associated DBMS, the platform on which the DBMS resides, and the network service used to reach that platform. The Data Source may be local, in which case there would be no network involved.

Figure 4.6 Basic Sequence of ODBC Function Calls



Connection information for each Data Source is stored in a special ODBC initialization file (ODBC.INI) which is created when the system is first configured, then updated as required. Administration software modules manipulate and control this file when needed.

All Drivers in the system support the following connection-oriented functions:

- SQLCONNECT enables an application to connect to a Data Source. The application provides identification data in the call to SQLCONNECT. This consists of the Data Source name, user ID, and an optional password.

- SQLALLOCENV allows the Driver to allocate storage for environment data.

- SQLALLOCONNECT allows the Driver to allocate storage for connection information.

When an application calls SQLCONNECT, the Driver Manager uses the Data Source name to determine the location of the Driver DLL. It then loads the Driver DLL and passes arguments from SQLCONNECT to it.

There may be additional steps involved, dependent on the nature of the Data Source. An error condition will be transmitted if the connection attempt is unsuccessful.

ODBC Software Installation

Users install ODBC software by implementing a vendor-supplied Setup program or one that is specific to the application. Setup calls an installer DLL in order to obtain information on the Drivers that accompany the package. The Setup program then installs all Drivers and associated files. The installer DLL receives information about each installed Driver at the end of the Setup procedure.

The complete installation system consists of the following components:

- Setup program and related files provided by the vendor, or a customized module appropriate for a specific application.

- An installer DLL that is part of the ODBC software development kit.

- An installation file which contains information about the Drivers.

- An installer DLL initialization file that records data about installed Drivers. This file is created by the installer during the Setup process.

Once the full configuration is installed, it may be modified as conditions change through use of an ODBC administration program. The latter enables addition or deletion of Data Sources via modification to installer DLL information.

ODBC Alternatives

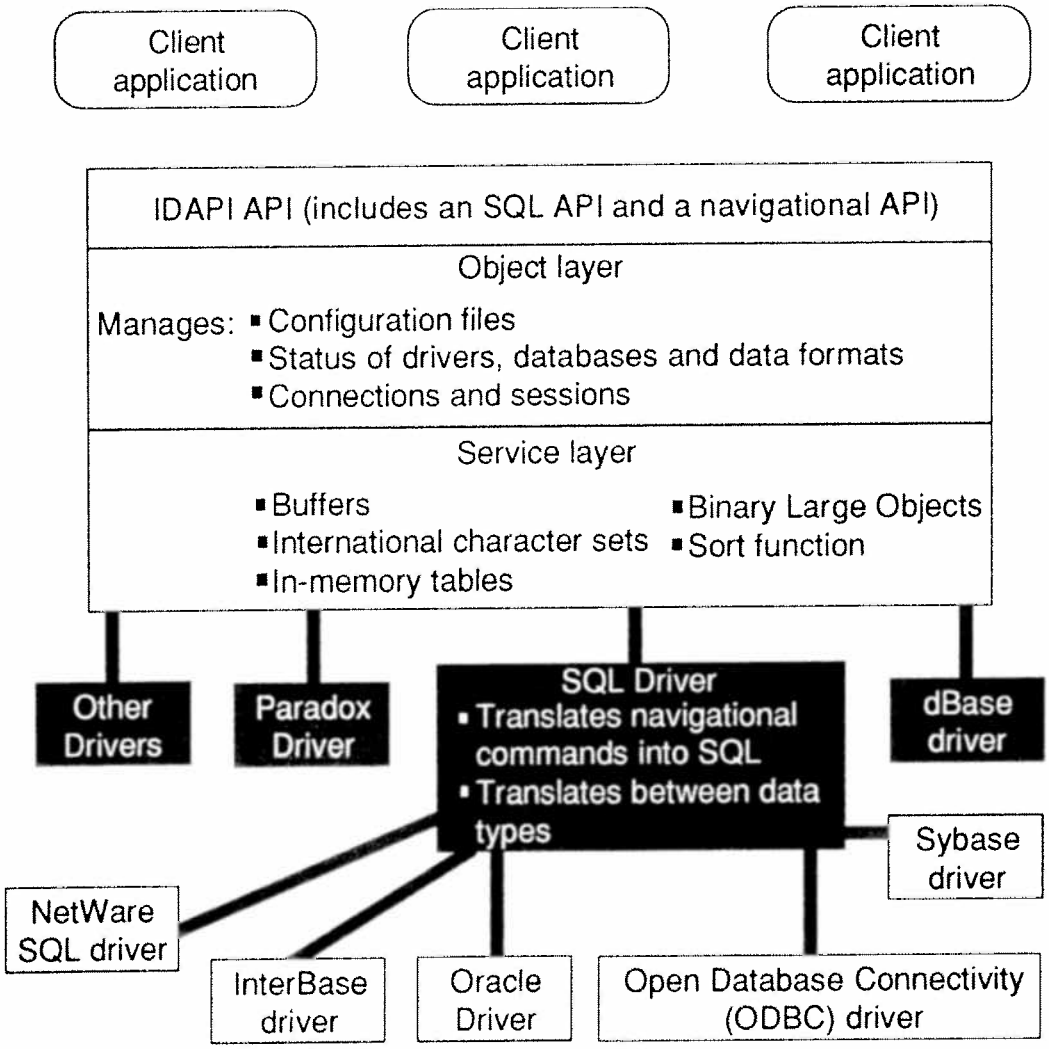
Although ODBC has received widespread industry support and may achieve de facto standard status, it focuses on the Windows environment. Other contenders have appeared which may diminish ODBC's influence over time. It is still too early to gauge their impact, however.

Oracle, for example, has announced an API offering called Glue. It will provide access to Oracle databases, as well as to data in electronic mail servers and the increasingly popular personal digital assistant devices.

Another group has produced an approach entitled Integrated Database API (IDAPI). Backed by Borland International, WordPerfect Corporation, Novell Incorporated, and IBM, it even supports ODBC. As with the latter, IDAPI is described as a superset of the SAG specification.

The IDAPI architecture (see Figure 4.7) consists of an API, an object layer, a service layer, an SQL driver, and various database drivers. It can also provide access to non-relational databases if so required.

Figure 4.7 IDAPI Architecture



The API consists of the SAG CLI for SQL users, as does ODBC. In addition, there is an IDAPI navigational API for programmers adept with record-oriented PC-type databases. Multiple API types allow users to access databases in a mode with which they are most comfortable.

The object layer is the central functional element of IDAPI. It converts function calls for eventual use by database drivers. It also loads database drivers as needed. The object layer manages multiple sessions between clients and database servers, including error control functions.

The service layer handles conditions specific to the client applications and database servers within the system. It also sets up buffers and performs related services.

Over 30 companies have committed to supporting IDAPI, some of whom also support ODBC. IDAPI developers claim their offering is complementary to ODBC, but emphasizes multivendor platform support in contrast to ODBC's Window-centric focus.

Chapter 5

Messaging API

History

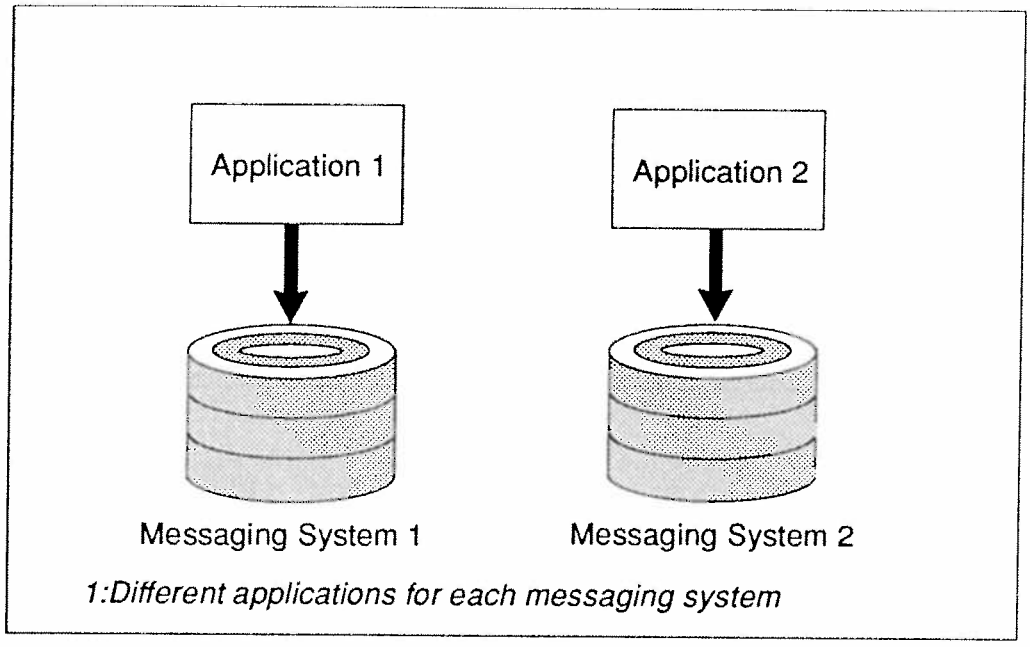
E-mail emerged in the 1960s on various computer timesharing systems as users sought new methods of communication. These early initiatives were primitive by today's standards, typically involving only a couple of day's programming effort. All implementations were, of course, unique to their own user group. Little thought was given to potential interoperability needs of the future.

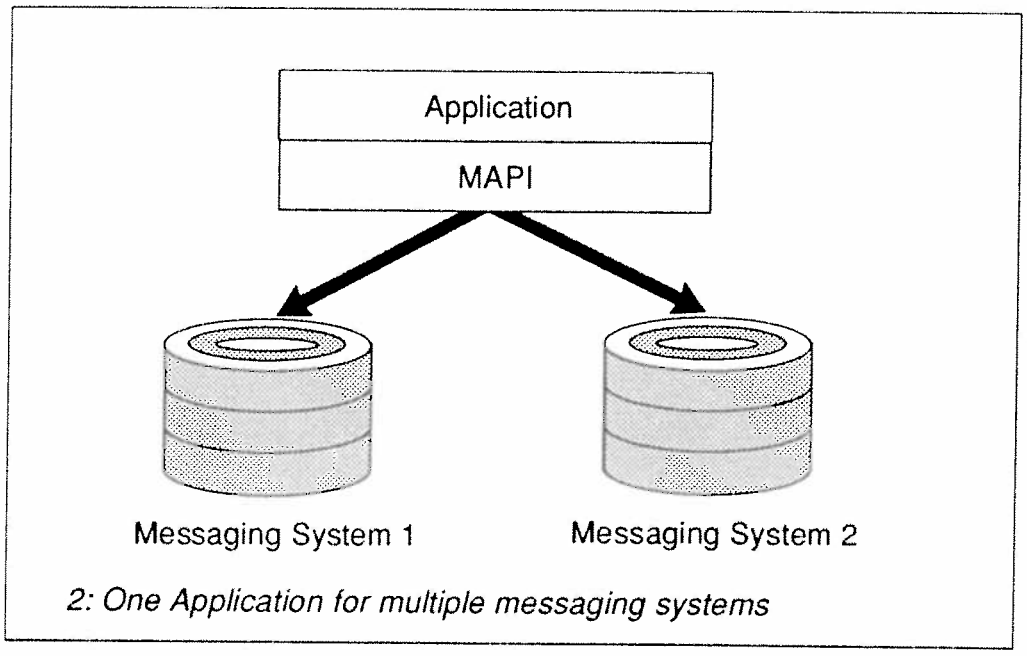
In 1969, the Advanced Research Projects Agency Network (ARPANET) was created by the U.S. government. It was designed to allow researchers within industry and academia to transmit computer data to each other, and to initiate program execution on remote systems. If one could send computer files, why not shorter messages such as E-mail?

ARPANET's prime contractor developed in 1970 a software vehicle that allowed their local mail system to communicate with independent mail systems at other ARPANET sites. This contractor, Bolt, Beranek, and Newman Incorporated, thus launched the network E-mail phenomenon. ARPANET users quickly became dedicated practitioners of messaging services. Commercial products gradually became available in the ensuing years.

MAPI is not E-mail; it is a facilitator of E-mail. In the years since ARPANET, users and other interested parties have searched for a universal access (or API) to the voluminous mail offerings that have appeared in the marketplace. If an application works with one product, it would be beneficial if that same application could send an identical message to another mail service without altering parameters in the sending application (see Figure 5.1).

Figure 5.1 MAPI Facilitator

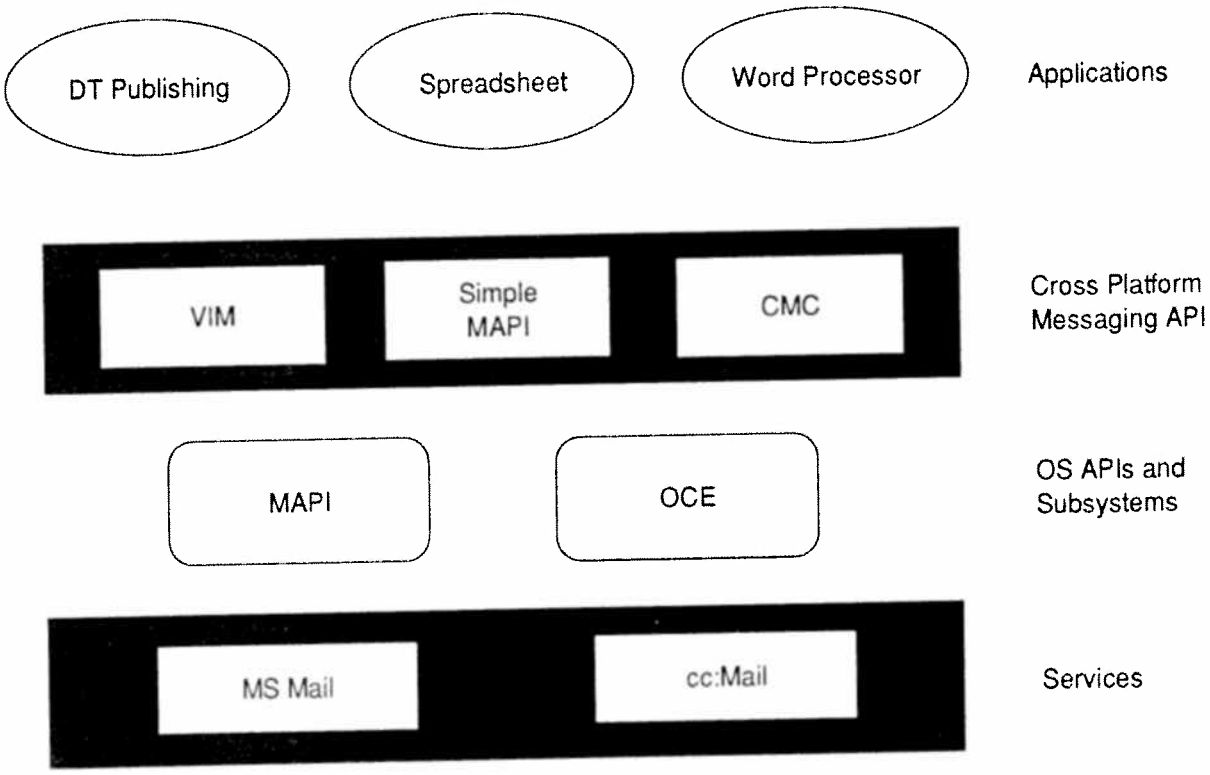




Source: Microsoft

This is what MAPI seeks to accomplish, albeit in a Microsoft Windows context. Others working toward the same goal, as shown in Figure 5.2, are the X.400 API Association with their Common Mail Calls, Lotus and others with VIM, Apple with Open Collaborative Environment, etc. See Appendix B for further details.

Figure 5.2 MAPI Vendors



VIM: Vendor-independent Messaging
CMC: Common Mail Calls
OCE: Open Collaborative Environment

Source: Microsoft

Despite all of these efforts, most current applications remain tied to messaging systems that are not interoperable with their peers. There is no commonality in the user interface, procedures for access, and supporting tools are unique to a particular platform or software module. Users attempting to implement workgroup computing find it difficult to succeed when even basic messaging systems are incompatible.

Thus, MAPI, VIM, and the others attempt to provide messaging applications with independence from various system anomalies. In fact, VIM is so determined to solve this problem that they have declared their intention to service MAPI environments as well as their own broad array of procedural accommodations.

The impact of solutions such as MAPI on enterprisewide message systems is profound. Developers can now add messaging capabilities to their applications without being concerned about the actual service provider. Instead of writing versions of their application for each messaging system, one MAPI-compliant version provides a capability for universal messaging.

In addition, workgroup computing becomes a more viable exercise. Everyone can communicate with each other. Data such as charts, reports, project schedules, and timesheets can be shared among a broad spectrum of corporate workers. Users of the Windows operating system all see the same graphical interface. The addition of MAPI to this scenario removes, at least to the user's view, the idiosyncrasies of underlying messaging systems.

Microsoft worked with dozens of vendors when formulating design parameters for MAPI. They all provided input toward the goal of producing an "open" messaging standard for industry, particularly for the Windows segment. A partial list of independent software vendors supporting MAPI appears in Table 5.1. Since MAPI may ultimately attain standard status, this list is only a fragment of support sure to emerge in the near future.

Table 5.1 A Sampling of ISVs Supporting MAPI

Forms Vendors	Beyond, Inc.
	Delrina Corporation
	JetForm Corporation
Word Processing Vendors	WordPerfect Corporation
	WordStar Inc.
Personal Information Management Vendors	FranklinQuest
	Polaris Software
Tape Backup Utility Software Vendors	Archive Software
	Fifth Generation Systems
Medical Software Vendors	PenKnowledge
Document Imaging Vendors	Alacrity Systems
	Keyfile Corporation
Network Utilities Vendors	Microcom
	Xtree Corporation
OCR Vendors	Calera Recognition Systems
FAX Modem Sharing Vendors	NUKO Information Systems
Database Vendors	Claris Corporation
	Pilot Software
	Software Publishing Corporation
Group Scheduling Vendors	Powercore, Inc.
	Raindrop Software
Office Management Vendors	Action Plus Software, Inc.
	Chronos Software
Electronic Meeting Vendors	GlobalStream Corporation
	Ventana Corporation
CAD Vendors	ISICAD, Inc.
Workflow-enabled Business Solutions Vendors	Dun & Bradstreet
E-mail Vendors	Capella Systems
	Lenel Systems
Dynamic Graphics Vendors	LABTECH
Wireless/Mobile Communications Vendors	Fourth Wave Technology, Inc.
Business Graphics	Shapeware

Among the many issues put forth by vendors cooperating in the MAPI effort were the following:

- Developers want open capabilities at both the client end, as represented by Windows, and also at the service end. Whether host or LAN-based, the proposed API must be able to deal with a variety of messaging systems. There must also be a uniformity of capabilities so that an application receives consistent messaging service.

- The messaging API should be integrated with the features of the underlying operating system. The operating system itself must include messaging functions and interfaces tailored to its unique strengths. By integrating functionality into the operating system, every user will have identical capabilities. If it was allowed to be an optional extra, the same users would not possess full functionality. In addition, developers can write advanced software knowing it impacts every user of the target operating system.

- Corporate-wide messaging requires an ability to communicate with alternate operating system platforms. Consequently, an operating system-based API must also support a cross-platform industry standard that provides, as a minimum, basic sending and receiving capabilities. MAPI addressed these needs by supporting the X.400 API Association industry-standard cross-platform effort. MAPI is also an integral component of the Windows operating system.

Messaging Subsystem

MAPI is more than just a set of standard APIs, although that alone is important enough to warrant attention. In Microsoft's vision, MAPI is a messaging subsystem present in the Windows operating system.

It works similar to their print drivers, i.e., different print drivers work in conjunction with Windows software, rather than directly with each application. This allows applications to function with a variety of printers. MAPI's messaging subsystem employs the same approach – different messaging applications can communicate with a variety of messaging services. MAPI is one of the first to integrate its services into the resident operating system.

When an API is initiated, a command or function call is invoked. It tells the underlying messaging subsystem what action is required. The messaging subsystem does not replace functionality provided by packages such as X.400, Novell's MHS, etc. Rather, it performs services that shield the user from the distinctness present in these packages.

In essence, the messaging subsystem performs the following tasks:

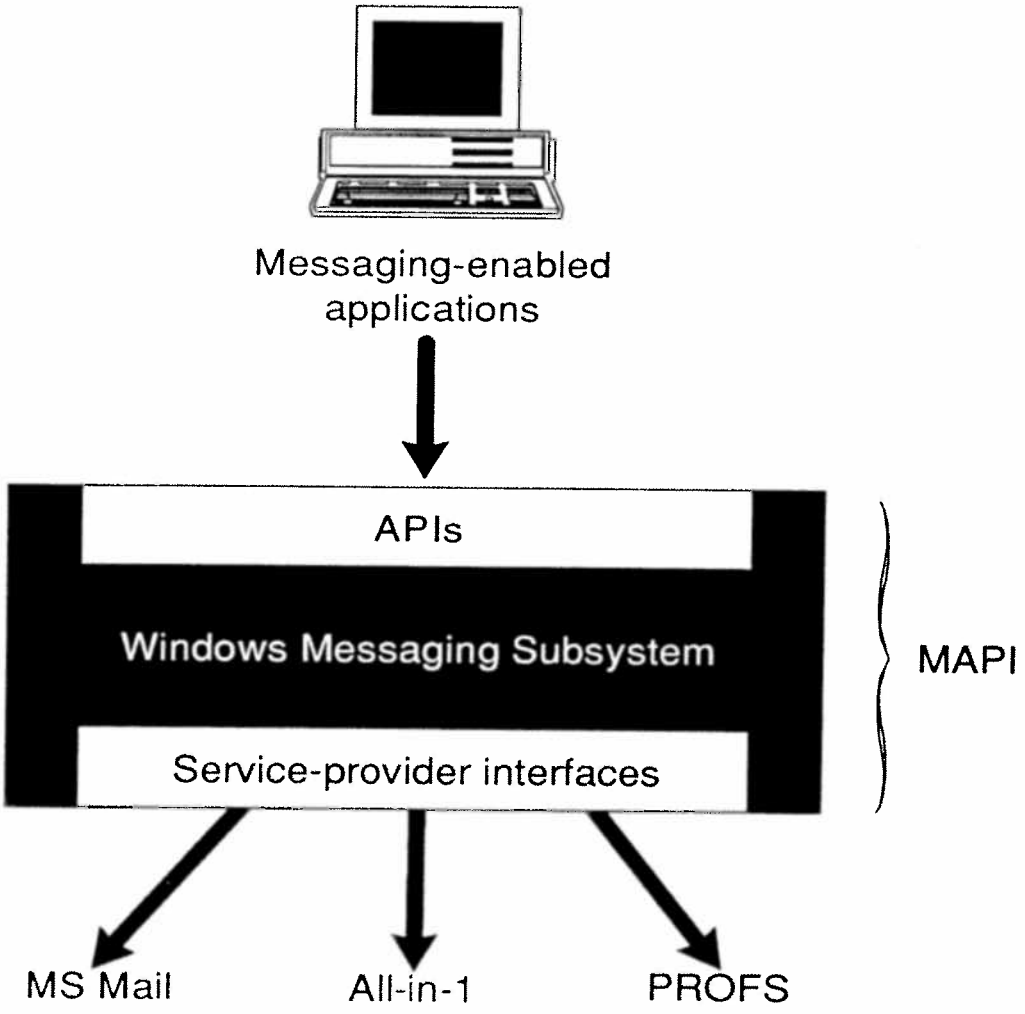
- provides common user interfaces for sending, receiving and storing messages;
- manages different message stores and address book directories;
- controls various communication transports needed to move messages among different messaging systems;
- holds messages for later transmission when a message system is disconnected;
- sends status information to applications when messaging activities occur.

Thus, software developers can create applications that work consistently for all users of the operating system. The same basic functionality will be present on every desktop. No accommodation has to be made for operating system users who may not have a piece of add-on software. The messaging subsystem is integral to that same operating system, not an implementation afterthought.

MAPI Architecture

It is no longer necessary, at least in a Windows environment, to create separate messaging and workgroup applications for each platform in a processing environment. As depicted in Figure 5.3, MAPI APIs service the application, MAPI SPIs accommodate different message system providers, and Windows messaging subsystem capabilities ensure that all users have the same tools.

Figure 5.3 MAPI Architecture



MAPI's set of function calls, in conjunction with the messaging subsystem, act as a broker between front-end (client) applications and back-end networks and messaging packages. This allows both application developers and messaging system vendors to be free from each other's concerns. They each can concentrate on improving the efficiency of their own products.

The MAPI front-end (client) APIs are available in two varieties. There is Simple MAPI for the most commonly used messaging tasks, and Extended MAPI for more advanced messaging services. Table 5.2 lists Simple MAPI function calls.

Table 5.2 MAPI Function Calls

MAPI call	Result
MAPILogon	Begins a MAPI session.
MAPILogoff	Ends a MAPI session.
MAPIFree	Frees the memory allocated by the messaging subsystem.
MAPISendMail	Sends a standard mail message. Messages can be sent without any user interaction or can be prompted via a common user interface (dialog box).
MAPISendDocuments	Sends a standard mail message. This call always prompts with a dialog box for the recipient's name and other sending options. It is primarily intended for use with a scripting language such as spreadsheet macro.
MAPIFindNext	Allows an application to enumerate messages of a given type. This call is targeted at incoming mail.
MAPIReadMail	Reads a mail message.
MAPISaveMail	Saves a mail message.
MAPIDeleteMail	Deletes a mail message.
MAPIAddress	Allows the user to create or modify a set of recipient entries using a common address dialog box.
MAPIDetails	Presents a dialog box that provides the details of a given address-book entry. The amount of information presented is determined by the address-book provider (on the back end) to which the entry belongs.
MAPIResolveName	Resolves a friendly name (an alias) to an address-book entry. This call offers the option of prompting the user to choose between ambiguous entries if necessary.

Simple MAPIs allow applications of various types operating in a Windows environment to perform messaging services. For example, a word processor or spreadsheet routine can forward data to targeted recipients. A scheduling utility can forward project milestones to workgroup members for their information and comment. Whatever the application, message initiators need not be concerned about the underlying messaging system or network entity in a MAPI-compliant configuration.

Extended MAPI provides more sophisticated service via additional APIs. These additional APIs are often unique to the installation. Applications employing Extended MAPI handle large and complex transmissions. They often require higher level addressing features due to the large volume of messages being serviced.

A timesheet form, for example, may be distributed periodically to collect labor records. Once this data has been appended, it would automatically be transmitted to a corporate payroll system for monitoring, editing, and payment action.

Extended MAPI's message store and address book capabilities offer the following services:

- *Message store* uses folders to organize messages. Folders contain messages which can contain attachments. Folders, messages, and attachments have parameters associated with them such as time sent and type. Folders are organized into a hierarchical, tree-structured format. Users can browse through this folder structure searching for a particular subject, type, sender, or whatever criteria is appropriate for the assembled body of information present. Received messages can also be modified, then returned to their folder for subsequent use.

- *Address books* are a collection of message recipient lists. Each list is called a container and can exist as a single

entity or in multiple form. If different messaging systems are used together, each with their own directories, MAPI presents a single master address book to a user that combines all of the directories available. This masks system complexities from that user.

Security features in a MAPI environment vary according to what is offered by the resident operating system. Windows NT, for example, provides more sophisticated security options than Windows 3.x. There are a variety of available procedures relative to user logons: from simple one-time user identification, to multiple, layered ID checks each time a user enters a new workgroup application area. The messaging subsystem encrypts all security data that it stores for a service provider.

MAPI uses object-oriented techniques for its messaging functions, although they do not necessarily follow any of the nebulous standards emerging in this arena. The MAPI object model is, however, consistent with Windows-type object-oriented models, recognizing all messaging system DLLs as separate objects.

As explained earlier, MAPI is integral to the Windows operating system. This enables it to offer advanced messaging, but diminishes its role as a multiplatform player. Microsoft is, however, an active participant in the X.400 API Association (XAPIA) which is creating a cross-platform API set. In addition, Simple MAPI will work with MS-DOS and Macintosh platforms during the interim while XAPIA completes its universal calls for basic cross-platform messaging functions.

MAPI Solutions

MAPI helps to solve three core problems endemic to any messaging system: supporting multiple messaging services with the same client, integration of services at the desktop, and working with specialized service providers.

Because MAPI removes the mutual reliance between client applications and the server messaging system, there is complete transport independence. This enables an organization to convert all users to the

same client E-mail application. Users will then have the same API on every desktop. Uniformity and consistency are achieved for the user interface.

The benefits of a common user interface extend to service integration at the desktop. Services such as facsimile, voice support, conventional mail, and links to third-party information utilities such as CompuServe can be accessed from the one, universal client application (see Figure 5.4). Appropriate Drivers would be installed for each type of service offered.

MAPI also more easily supports various service providers such as database packages, corporate directories, and related entities due to its modular configuration. Address books, message stores, transport mechanisms are all supported separately. Thus, an organization can select whatever back-end packages that are needed, while maintaining the aforementioned universal client front-end application.

Alternative Standards

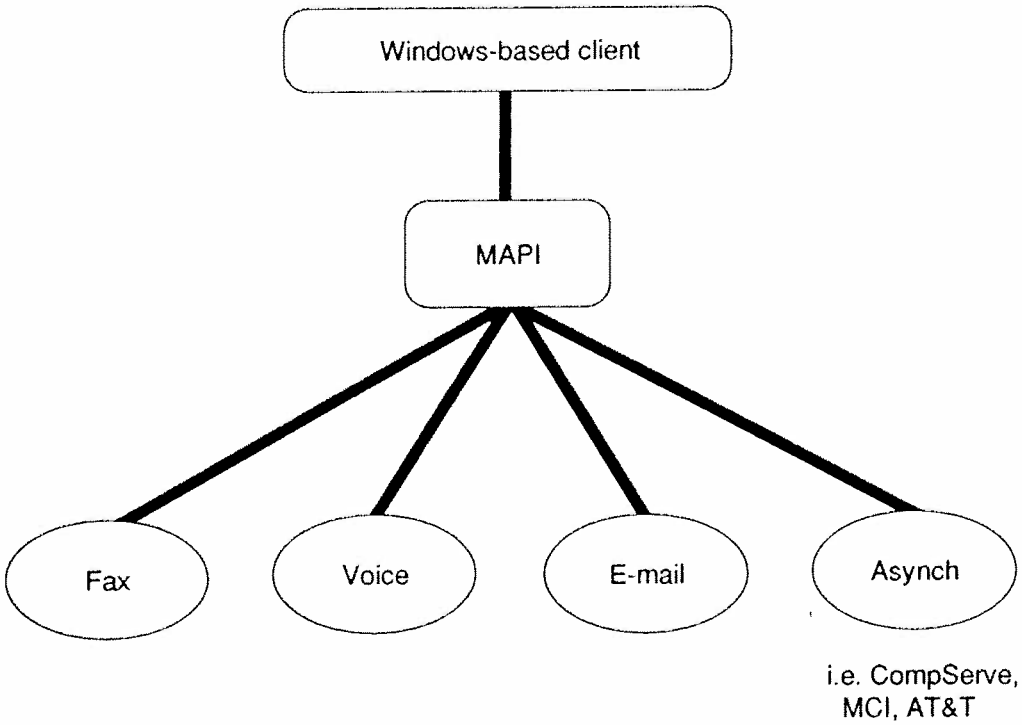
Microsoft is a member of XAPIA, the cross-platform messaging API that is based on the X.400 standard. It has declared its intent to support XAPIA's Common Mail Calls (CMCs) which are similar in functionality to Simple MAPI capabilities.

Microsoft also supports Apple's Open Collaborative Environment (OCE) for Macintosh environments. These, along with VIM, are the major alternative API standards of direct interest to MAPI developers and practitioners at this time. VIM has declared its intent to support MAPI calls, so there is a degree of confluence in these multiple efforts.

Future Directions

MAPI is a full-featured messaging system. It provides multiple service interfaces (transport, address book, message store). Each enables developers to create installable DLLs for their services. The SPI mechanisms allow an application to access multiple transports, address books, and message stores. MAPI also offers an object model and features full integration into Windows.

Figure 5.4 Integration of Services



MAPI's kinship with Windows is both a strength and a weakness. It is a weakness in the sense that it is Windows-centric. Whether it can be successfully ported to foreign environments and still maintain full functionality is yet to be determined.

On the other hand, Windows integration means MAPI is universal in the marketplace. Everyone has identical capabilities on the desktop. Vendors and users will not have to worry whether all components are present. They will have a uniform presence.

This uniform presence will aid MAPI in addressing future user concerns, particularly in areas such as groupware and mail-enabled application support. With the latter, functions such as word processors and spreadsheets are becoming mail-enabled, which allows data to be forwarded directly into the mail network.

APIs such as MAPI, VIM, OCE, and XAPIA are encouraging the development of yet more mail-enabled and groupware applications. They then use the mail network as a transport capability between applications. Within the next two to three years, API usage within groupware environments will also greatly enhance this growing segment of the market.

There is an excellent opportunity for MAPI to achieve standard status due to the strength of Windows. The other efforts all have their strengths, however, and will continue to serve those environments from which they emerged.

Chapter 6

License Service API

The arrival of PCs into corporate environs bestowed great benefits on users, but also opened up the problem of software licensing. Generally, to help control software usage, vendors ask purchasers to sign a license agreement and then enforce the agreement through various methodologies.

There are three such methodologies that apply to PC software. It can be licensed: 1) to the individual, 2) to the machine, or 3) for concurrent use.

An individual license means, as the name implies, that the software is licensed to a specific individual. For example, if a network has 30 PCs, but only 20 of its users actually run the software in question, then 20 licenses are required, one for each user. Problems relating to movement of the user, etc. are handled uniquely by different licenses.

When software is licensed to a machine, that software is authorized for use on a specific device. If a network with 30 PCs has 20 machines with the software installed, then 20 licenses must be in place in order to maintain legal propriety. If a user decides to install that software on a portable device, for example, it must be erased from one of the existing PCs, or another license must be purchased.

With a concurrent use type license, the determining factor is the number of copies of a software product that are in use at any given point in time. For the 30 PC network with 20 users example, if only 15 of the 20 users run the software at the same time, then 15 licenses are required, not the 20 licenses mandated by the two previous license types.

If concurrent usage of the software package exceeds the original number at some time in the future, additional licenses must be acquired. This type of license demands more management than other types. An installation may employ a third-party software routine to monitor software package concurrent use. Many vendors implement lock-out mechanisms as a means to control user access. This software utility allows only a set number of users to access a program at any one time.

The increasing use of portable and home-based PCs has led to development of another licensing approach. Some allow only one copy to be memory-resident at one time. Other licenses impose a percentage division of software usage. For example, 75% on a designated primary machine and 25% on a secondary machine such as a laptop device.

Licenses are often maintained by a license server that supports one or more licensing procedures. Each license server is available to several network stations. Since each software vendor can select its own licensing terms, license servers must be able to deal with various licensing models. For example, concurrent, individual, and specific machine licensing may all be encountered in a random configuration.

There are several organizations involved in software licensing, monitoring and enforcement. The Software Publishers Association (SPA) is a large trade association for the PC software market. It has over 1,000 members and vigorously battles software piracy in North America and Europe.

The Business Software Alliance (BSA) is another influential consortium of software companies determined to enforce copyright laws on an international scale. Key members include Microsoft, Novell, Aldus, WordPerfect, Autodesk and Lotus. Most of these firms are also members of the SPA. The BSA has been particularly active in the Orient and third-world countries.

There is a third organization headquartered in England entitled the Federation Against Software Theft (FAST). Its efforts are primarily focused in the U.K.

LSAPI Goals

The LSAPI specification is a vehicle which software publishers can use to incorporate license verification into their products, in a manner independent of a particular software licensing system. This API allows vendors to create a single approach to all licensing systems that support LSAPI conventions.

LSAPI minimizes vendor difficulty in implementing software licensing, potentially requiring only two function calls to accomplish the task. When using LSAPI, the application does not need to know anything about supporting infrastructure issues such as type of computers in use, or type of network. Instead, the application can connect to all licensing services needed across multiple computing environments in a platform-independent manner.

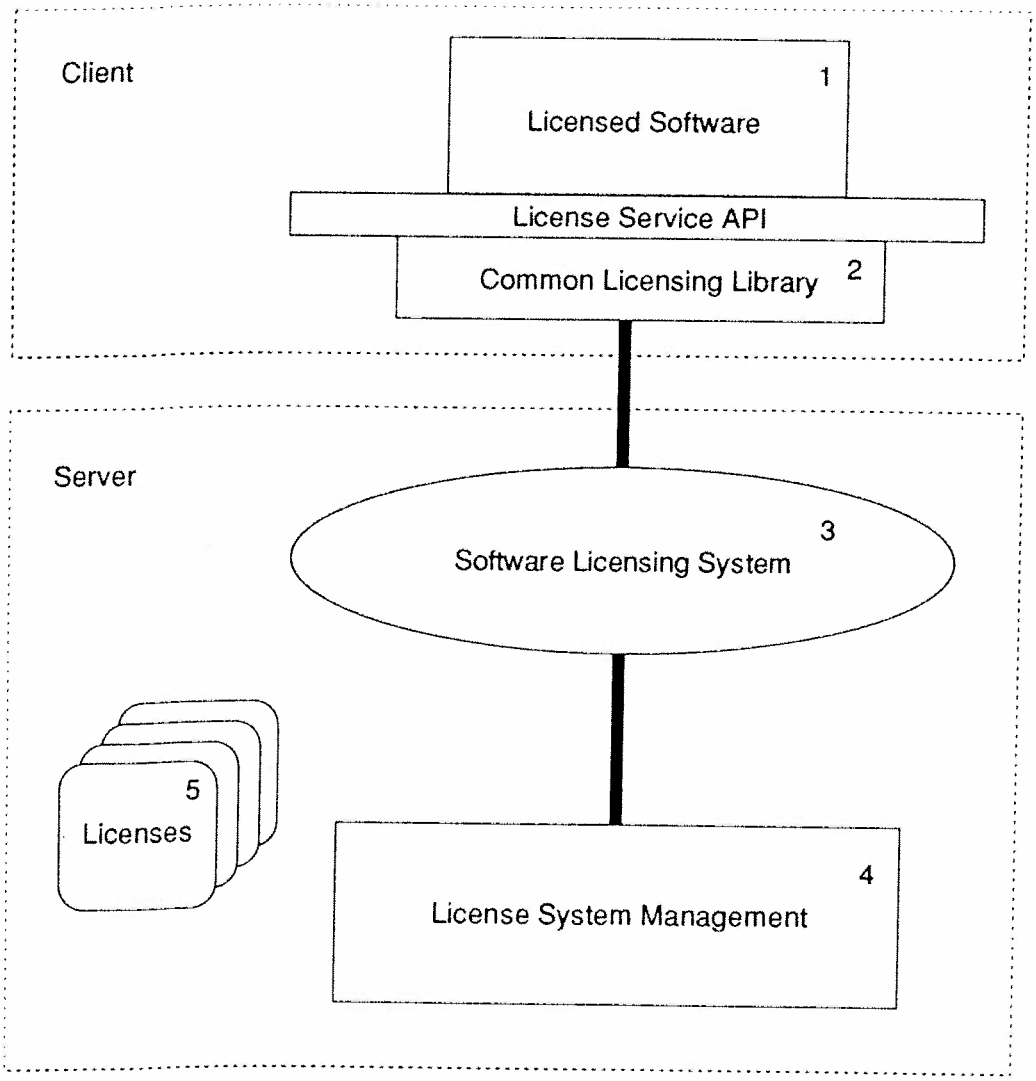
Figure 6.1 depicts a software licensing model in its totality. It also shows the model as it would operate in a client/server, networked environment. In a single system situation, both server and client entities could reside on the same platform. LSAPI deals solely with the interface between 1 and 2 in the Figure. Components numbered 3, 4, and 5 deal with the licensing system itself, along with its management, terms, and conditions.

The specific goals of LSAPI are many, but perhaps the most fundamental is: Help the honest person remain honest. Other more precise goals are:

- Provide a relatively simple API which software developers can use for licensing. It should be appropriate for a diverse audience of software publishers, licensing schemes, operating systems, and network configurations.

- Support application interfaces to multiple license systems.

Figure 6.1 Software Licensing Model



Source: Microsoft

- Provide an API that not only supports, but actually facilitates, interaction with licensing methodologies such as concurrent use, individual use, or machine-based.

- Minimize the effort required to implement use of software licensing within a vendor's products.

- Allow software developers to isolate their product's program code from specific licensing policy. The latter will be managed by the target licensing system.

- Be scalable to operate on a full range of processing platforms, from the smallest to the largest, networked or single system.

- Provide security controls against "meter adjustments," i.e. prevent tampering with the licensing system.

It should be emphasized that the goals of LSAPI are focused within boxes 1 and 2 in Figure 6.1. Issues, of which there are many, associated with the rest of the licensing model are beyond the scope of LSAPI.

LSAPI Function Calls

The following calls support basic functions of the licensing system. They include the ability to request the licensing system to grant application software permission to run, to terminate that permission when no longer needed, and to modify the status of licensing resources granted to the requesting software product.

Each LSAPI function call is accompanied by arguments appropriate to the command being executed. At present, there are six LSAPI calls:

1) *LSRequest* – Requests the licensing system to grant the calling software authorization to execute. Among the arguments are license system identifier, name of the product requesting license resources, and name of the publisher.

2) *LSRelease* – Requests that the licensing system release licensing resources associated with a specific procedure.

3) *LSUpdate* – Updates synchronization between licensed software and the licensing system, including previous usage of the software.

4) *LSGetMessage* – Returns the message associated with LSAPI status code, such as error determinations.

5) *LSQueryLicense* – Obtains information about the license or service provider, such as what type of license type is applicable.

6) *LSEnumProviders* – Identifies the vendor, product, and licensing system version of the installed license system providers.

LSAPI Environments

LSAPI was created to function successfully on many operating system platforms. Windows is obviously a primary target, but others can utilize this package. Operating environments which support dynamic linking have a distinct advantage. The LSAPI architecture allows multiple, concurrent license system service providers when dynamic linking is supported. In addition, application programs are not bound to a specific license system.

Operating systems which do not have dynamic linking, such as MS-DOS, may invoke emulation procedures to achieve support for multiple, concurrent service providers. If dynamic linking is not supported in any manner, then the application is statically linked to a vendor-specific licensing module as illustrated in Figure 6.2.

Figure 6.2 Static Linking

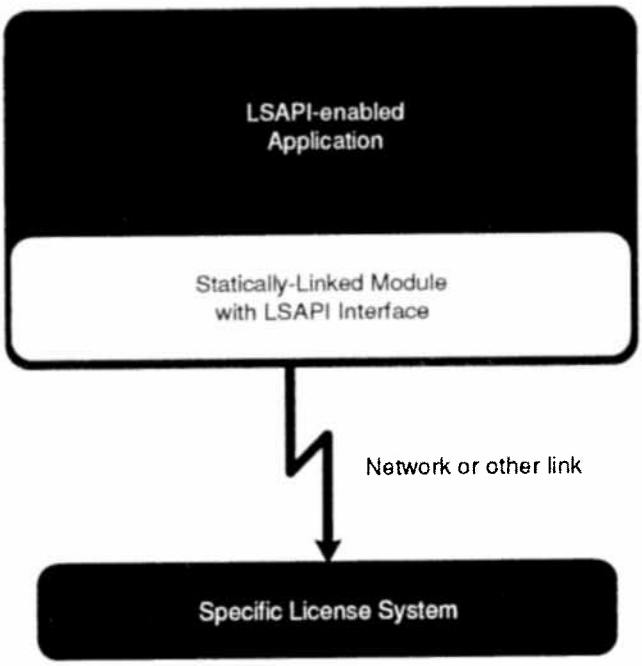


Table 6.1 Sampling of Operating Systems with LSAPI-related Characteristics

	Source Code License Independent?	Executable Code License System Independent?	Dynamic Linking Native to Operating System?
Windows	Yes	Yes	Yes
MS-DOS	Yes	Yes	No*
Macintosh	Yes	No*	No*
OS/2	Yes	Yes	Yes
Open VMS	Yes	Yes	Yes
Unix	Yes	No**	No*

* Runtime license system independence may be provided by a library which is statically linked to the application.
** Some Unix versions may provide for runtime linking.

LSAPI Benefits

Perhaps the major disadvantage of having many non-standard licensing systems is the burden placed on users to learn about each system and which applications interface with which. If additional licensing systems are added, more must be understood about the multiplicity of systems. Requiring people to absorb all this peripheral information is looked upon as an impediment to productivity. To the user and network administrator, licensing should proceed automatically and transparently, with little or no impact on their daily tasks.

The benefits of LSAPI, therefore, are important in terms of overall productivity. By providing a single, system-level interface for connecting front-end applications with back-end license services, software developers and users alike do not have to worry about conversing with numerous services, each with its own protocols and APIs. Making these connections becomes the job of license system providers and the operating system management layer, not of individual applications.

The License Service API provides a framework in which licensing-enabled applications can seamlessly access license information in a distributed computing environment and on a stand-alone PC. It accomplishes this by making a common set of APIs available to all applications.

Like two diplomats conversing through an interpreter, a front-end application and back-end service do not need to know how to speak each other's language in order to communicate, as long as they communicate through the License Service API. As a result, LSAPI allows applications developers, information systems managers, and vendors of back-end license services to mix and match applications and services to build enterprise solutions that shield programmers and users from the underlying complexity of the license system.

Although the primary benefit of LSAPI is its ability to provide software publishers with seamless connection to license systems, other significant benefits include:

- Easier license management for IS managers. Because LSAPI simplifies the software licensing process, it enables IS managers and network administrators to more easily track application usage. Through proper tracking and control, the terms of the license agreement can be properly maintained.

- Easier upgrade paths for IS managers. Because LSAPI enables a single application to work with multiple back-end services, IS managers can upgrade or change their licensing services without affecting the users or their applications. Each set of users can use the software that suits their needs, and IS managers can deploy solutions without fear of interfering with strategies planned for the future.

- Expanded market opportunities for software publishers. Adding support for new types of license systems can be extremely costly, and a publisher's potential market might be limited to those organizations that use the license systems already supported by the publisher's product. With the License Service API, the software publisher's market opportunities are expanded to include customers who use any license service that supports LSAPI.

- Reduced costs for software publishers. Embedding license management and metering into an application is an arduous process. LSAPI relieves both corporate developers and software publishers of this burden by providing a single interface for all applications. Through this single point of access to different back-end license services, LSAPI eliminates the need for applications developers to rewrite their applications for each new licensing policy.

- Reduced burden on licensing service providers. With a standard interface to access back-end license services,

LSAPI reduces the burden on the back-end service vendors to provide system software and interfaces to their products. Because new implementations of back-end services can be accessed from a common API, the vendors need only include the service provider interface library for their products.

- Platform independence. LSAPI is not tied to any environment. It can be implemented within the Windows operating system, VMS, Unix, Macintosh, and OS/2. Support for the API within multiple environments allows software vendors to port their applications to different operating systems without rewriting the licensing portion of their software.

- Network transparency. LSAPI is designed to isolate applications from the details of network communication with a license server. It is the responsibility of each license system service provider to handle the protocol details. As a result, the underlying network in any environment becomes incidental to the applications developer. Networks such as Novell NetWare, Microsoft LAN Manager, DEC PathWorks, and Banyan VINES can provide support for LSAPI.

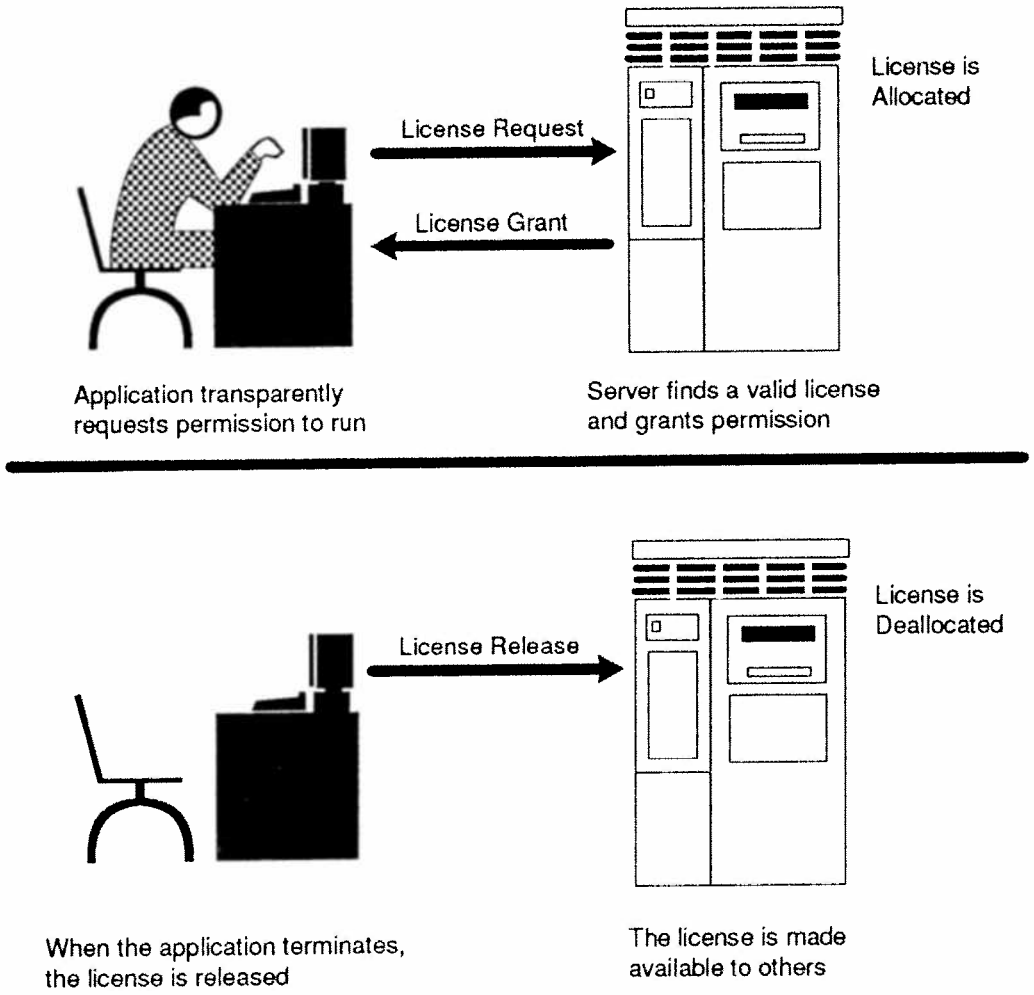
As license service vendors begin implementing support for the License Service API within their products, applications written to the LSAPI specification will be able to access an increasing number of licensing systems. If these license services are upgraded, or if new license service products appear, the same applications will be able to immediately access them without modification, as long as the license services comply with the specification. The software publisher need only issue a license appropriate for the license system in use.

Operational Overview

Under the LSAPI model, an application must request licensing services from a license server each time the application is started. Figure 6.3

illustrates a typical licensing process of a License Service API-enabled application.

Figure 6.3 How an Application Requests a License from a License Server



To ensure that the system is reasonably tamper-free, the request/grant process is authenticated through the use of "secrets" that are exchanged by the client application and the license provider. Chosen by the software publisher, these secrets are typically encrypted within the license itself, and only the license provider knows how to decrypt them.

The secret authenticates the license, providing protection against attempts to impersonate a license service.

In Chapter 1, Figure 1.7 shows the implementation of LSAPI within the Windows environment. This implementation of the API is integrated into the larger framework of WOSA, which provides even further independence of the application from the underlying system.

As the figure illustrates, a license-enabled application first calls functions that are part of LSAPI. Next, LSAPI functions interact with a provider management layer within the operating system that manages connections to multiple license service providers, among other tasks. The appropriate service provider interface then "translates" LSAPI function calls into the license service provider's native function calls. Once in native format, the information is sent to the appropriate license service for processing.

In the Figure 1.7 environment, LSAPI is independent of the particular licensing system or licensing policy and allows multiple license providers to transparently coexist. In other words, licensing software on a server can implement a wide variety of licensing schemes, yet still remain independent of LSAPI. The License Service API only provides a standard means through which applications and license service providers communicate – it does not specify the licensing mechanism or policy. Because of this, software publishers are able to choose the licensing system and policy that best suits their business needs.

LSAPI Vendor Support

The job of defining LSAPI was initiated by Brightwork Development, DEC, Gradient Technologies, Microsoft, and Novell. Many other vendors representing diverse market segments have joined the effort and expressed their support as seen in Table 6.2.

Table 6.2 Vendors Who Support the Job of Defining LSAPI

Apple Computer Inc.	Micrografx Inc.
Banyan Systems Inc.	Microsoft Corp.
Brightwork Development	NetWare Masters Group
CompuServe Inc.	Novell Inc.
DEC	Open Software Foundation
Funk Software	Oracle Corp.
Gradient Technologies Inc.	Software Publishers Association
Hewlett-Packard	Symantec Inc.
Highland Software	Tangram Systems Corp.
Lotus Development Corp.	WordPerfect Corp.
Microcomputer Managers Association	XTree

Conclusion

The primary goal of LSAPI is to make it easy for software publishers to integrate personal computers with a wide variety of licensing services. This enables software publishers and systems vendors to focus on building powerful solutions rather than overcoming incompatibilities between various products.

Corporate IS groups also benefit from LSAPI. Having a standard means of accessing multiple licensing services means that software licensing is simplified across a wide range of platforms and allows IS managers to easily change or upgrade applications or license services. As a result, corporate IS personnel can worry less about software licensing and focus more on managing their computing resources.

Chapter 7

Windows Sockets API

The specification for Windows Sockets (WinSock) defines a network programming interface for Windows systems which derives its inspiration from the University of California's BSD 4.3 Unix offering. It contains both traditional Socket routines and extensions designed to exploit Windows' message-driven nature.

Early versions of WinSock are targeted for use in TCP/IP environments. Using the API with alternate protocol suites is slated for future releases. In TCP/IP environments, it provides a high level of familiarity for programmers accustomed to using Sockets in conventional Unix-TCP/IP venues.

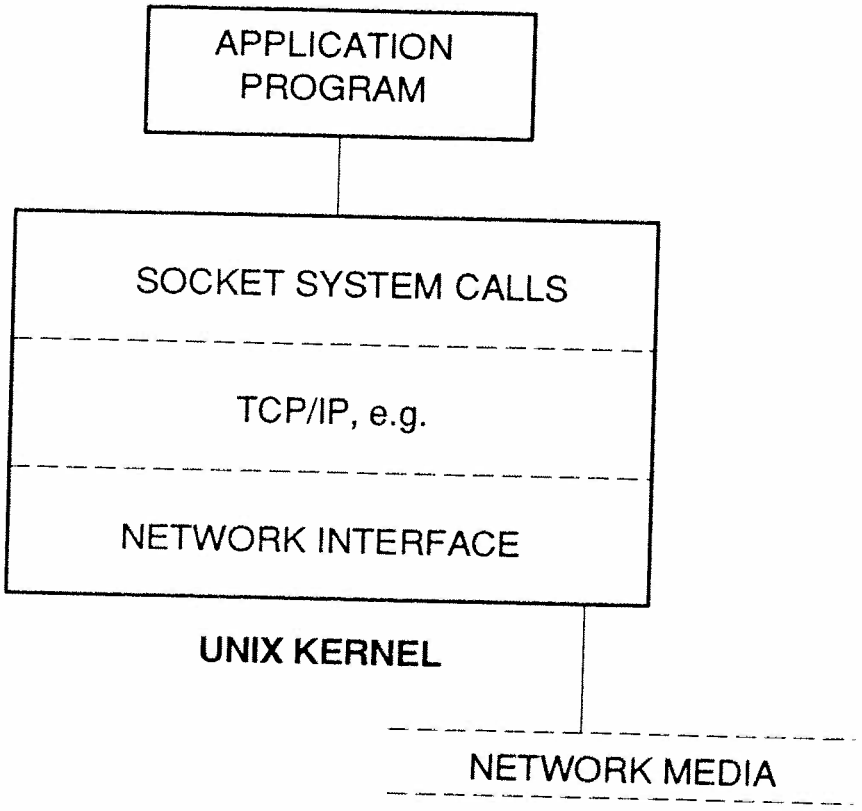
This API operates with all versions of the Windows operating system from 3.0 onwards. In this context, it can function with both 16- and 32-bit configurations. The aforementioned extensions in WinSock enable application developers to generate software that conforms specifically to the Windows programming model. They also aid in creating more robust applications with their expanded capabilities.

The origin of the WinSock API traces back to 1991. An ad hoc committee was established at a computer conference to develop a specification for Sockets in the Windows environment. By early 1992 the initial draft document was published. Early leaders in the specification development effort were JSB Corp., NetManage Inc., FTP Software, and Microsoft itself: Additional contributions came from individuals in companies or organizations such as Sun Microsystems, Massachusetts Institute of Technology (MIT), Hewlett-Packard, and 3Com.

Sockets Basics

Sockets, which first appeared in BSD Unix 4.2, are communication end points to which names can be attached. They serve as APIs between user applications and network protocols. TCP/IP is the predominant implementation. Most Unix products now support Sockets and its emerging competitive API: System V Transport Layer Interface (TLI). Figure 7.1 depicts the Socket system call entity within the Unix kernel.

Figure 7.1 Socket System Call Interface



Several system calls allow users to access the transport service. The major calls are listed in Table 7.1.

Table 7.1 Principal Socket System Calls

System Call	Service
Socket	Create a Socket of a specific type
Bind	Provide a name for an existing Socket
Close	Terminate the use of a Socket
Listen	Create a queue for incoming connection requests
Accept	Access a request or wait for one
Connect	Invoke a connection with a remote Socket
Send	Send data on a specific Socket
Send To	Connectionless version of Send
Recv	Receive data on a specific Socket
Recv From	Connectionless version of Recv
GetPeerName	Obtain address of peer process connected to a Socket
GetSocketName	Obtain local address associated with a Socket

A summary of their functions follows:

- *Socket.* To initiate network I/O, a user process must implement this command. It contains three arguments: protocol family (e.g., the Internet, Appletalk network, etc.), communication type (e.g., connection-oriented or connectionless), and specific protocol (e.g., TCP/IP). Not all combinations of these arguments are valid. For example, a protocol family might not support all communication types.

- *Close.* When a process is finished with a Socket, it issues this command. A single argument identifies the Socket to be closed. If a process aborts unexpectedly, most systems automatically close all open Sockets.

- *Listen.* Once a Socket has been created, buffers can be allocated to store multiple incoming connection requests. This is initiated by the Listen command. It contains two arguments: socket (identify the Socket that will receive connections), and queue length (specify the length of the request queue for the Socket). Listen is used only by connection-oriented servers for stacking service requests.

- *Bind.* Names are attached to the Socket by this command. Once assigned, a name becomes available to a remote process, perhaps a server, so that it can address the Socket. It contains three arguments: socket (identify the Socket to be bound), local address (specify the local address) and address length (number of bytes in the address). Not all argument combinations are valid. For example, a local address may already be in use by another program.

- *Accept.* Once a Socket has been created, the server takes the first connection request on the queue, if any. Otherwise, it awaits such a request. The Accept system call initiates this process. It has three arguments: 1) socket (identifies the Socket on which to wait), 2) address (becomes the address of the client placing request), and 3) address length (provides the length of the client address).

- *Connect.* A Socket is initially not associated with a remote destination. Connect links an application program to a specific Socket. This is typically a client connecting to a specific server. There are three arguments: 1) socket (identifies the Socket to connect), 2) destination address (specify Socket address), and 3) address length (provides the length of the destination address).

- *Send.* There are several variations for this system call, depending on whether there is a connection-oriented or connectionless Socket. Send works with the former. Send To is an example of the connectionless variety. The differences between them are trivial. The purpose of both is to transmit data via a Socket. Included among the arguments are: socket (specifically the Socket to use), message (provides the address of the data to be sent), length (identifies the length of the data to be sent), and flags (contains control information relative to data transmission).

- *Recv.* As with Send, there are several variations. Processes call Recv to obtain data from a connected Socket. It has four arguments: socket (specifies Socket from which data should be received), buffer (identifies where to place the data in memory), length (provides the length of the buffer area), and flags (contains control information in order to manage the transmission). Recv From is an example of a call to a connectionless Socket.

- *Get Peer Name.* Newly-created processes inherit open Sockets from the process that created them. In order to obtain the address of a peer to which a Socket connects, this system call is issued. It has three arguments: socket (specify the Socket for which the address is desired), destination address (points to the location to receive the Socket address), and address length (points to the integer that receives the address length).

- *Get Sock Name.* Returns the local address associated with a Socket. It has three arguments: 1) socket (specifies the Socket for which the local address is desired), 2) local address (points to the location that will contain the address), and 3) address length (points to the integer that will contain the address length).

In addition to the conventional activity of binding and using Sockets, application programs may wish to impose other conditions relative to Socket usage. Examples of this are buffer space management, send broadcast messages, retransmit parameters, etc. There are two system calls available that enable the user to implement processing options: Getsockopt and Setsockopt.

Getsockopt allows the application to obtain information about the Socket. It has five arguments:

1) socket, which identifies the Socket for which information is needed;

2) level, which specifies whether this activity applies to a Socket or its underlying protocol;

3) option ID, which specifies a single option to which the request applies;

4) option value, which points to the buffer address into which the requested value is placed;

5) length, which points to the integer address into which the option value length is placed.

Setsockopt permits an application program to set a Socket option using identical values obtained with the previous system call. It has the same five arguments as Getsockopt, except the length entry will contain the length of the option passed to the system.

Not all options apply to all Sockets. Variables such as underlying protocol and Socket status impact the specific choice of options used.

Sockets as implemented in BSD Unix are an integral part of the operating system kernel (see Figure 7.1). That feature is both its strength and its weakness. It is positive in that performance is enhanced by a tight coupling of Socket function and Unix kernel

processing. There are no intermediate layers or steps incurred when a Socket system call is invoked.

It is negative in terms of adaptability. Changes to Socket functionality or underlying protocol stacks requires modifying the core of the operating system – the kernel. This is the antithesis of openness and non-modularity, and is the downside of Socket integration into the kernel. The other API to the transport protocols (TLI) ameliorates this close-knit protocol association and allows broader protocol interaction.

Berkeley Deviations

As cited earlier, WinSock adheres closely to the BSD Unix implementation of Sockets. There are some deviations that have occurred, however, due to the unique requirements of a Windows environment. They include the following:

- A new data type (Socket) has been defined in order to anticipate file needs in future Windows releases such as NT. It will also aid in porting applications from 16-bit to 32-bit configurations.

- Error codes set by WinSock are handled differently than in BSD implementations. The latter makes them available by the global "errno" variable. An API is utilized in WinSock which makes it more adaptable to future Windows evolutions.

- All pointers used by WinSock applications are of one specific type. To facilitate this, new data type definitions have been provided.

- Some BSD function names have been altered due to duplication with Windows API names.

- WinSock uses a special command to close Sockets since, unlike BSD connections, Socket descriptors do not always correspond to their file counterparts.

- The maximum number of Sockets supported by any vendor's product is implementation specific. A typical number is 64, but this can be varied at compile time by changing appropriate parameters.

Windows Extensions

WinSock offers a number of extensions (see Table 7.2) to the basic set of BSD Socket routines. These extended APIs deal primarily with message-based, asynchronous access to network operations. Use of these extensions, when appropriate, is recommended for conformance with the Windows programming model.

Implementation

At last count, membership in the Windows Sockets development group exceeded 300 in number. They come from a broad range of organizations, companies, and agencies.

Many of these organizations have tested commercial implementations of Version 1.1 of the WinSock specification. Among those testing successfully are firms such as IBM, FTP Software, 3Com, Microsoft, NetManage, JSB, Novell, SunSelect, Ungermann-Bass, and The Wollongong Group.

Future developments will focus on achieving standards recognition for WinSock, particularly in the Internet community with their "Request For Comments" documents. Additional commercial implementations can also be expected as the standard matures.

Table 7.2 Windows Sockets Extensions

WSAAsyncGetHostByAddr()	A Set of functions which provide asynchronous versions of the standard Berkeley **getXbyY functions**. For example, the **WSAAsyncGetHostByName** function provides an asynchronous message-based implementation of the standard Berkeley **gethostbyname** function.
WSAAsyncGetHostByName()	
WSAAsyncGetProtoByName()	
WSAAsyncGetProtoByNumber()	
WSAAsyncGetServByName()	
WSAAsyncGetServByPort()	
WSAAsyncSelect()	Perform asynchronous version of **select**
WSACancelAsyncRequest()	Cancel an outstanding instance of a **WSAAsyncGetXByY** function.
WSACancelBlockingCall()	Cancel an outstanding "blocking" API call
WSACleanup()	Sign off from the underlying Windows Sockets DLL
WSAGetLastError()	Obtain details of last Windows Sockets API error
WSAIsBlocking()	Determine if the underlying Windows Sockets DLL is already blocking an existing call for this thread
WSASetBlockingHook()	"Hook" the blocking method used by the underlying Windows Sockets implementation
WSASetLastError()	Set the error to be returned by a subsequent **WSA GetLastError**
WSAStartup()	Initialize the underlying Windows Sockets DLL
WSAUnhookBlockingHook()	Restore the original blocking function

Chapter 8

Windows SNA API

SNA API Overview

In view of Microsoft's determination to further penetrate the enterprise and IBM's goal of seeking a wider platform base for SNA, the emergence of this API could have been predicted. IBM, Microsoft, and 18 other companies listed in Table 1.2 jointly cooperated in developing the specification.

An application written to these standard interfaces will run unchanged over many vendors' connectivity products, operating under both Windows and Windows NT. Currently, Windows applications accessing SNA are linked to a particular vendor's SNA interface software. Change the connectivity product, and the application will also require alteration.

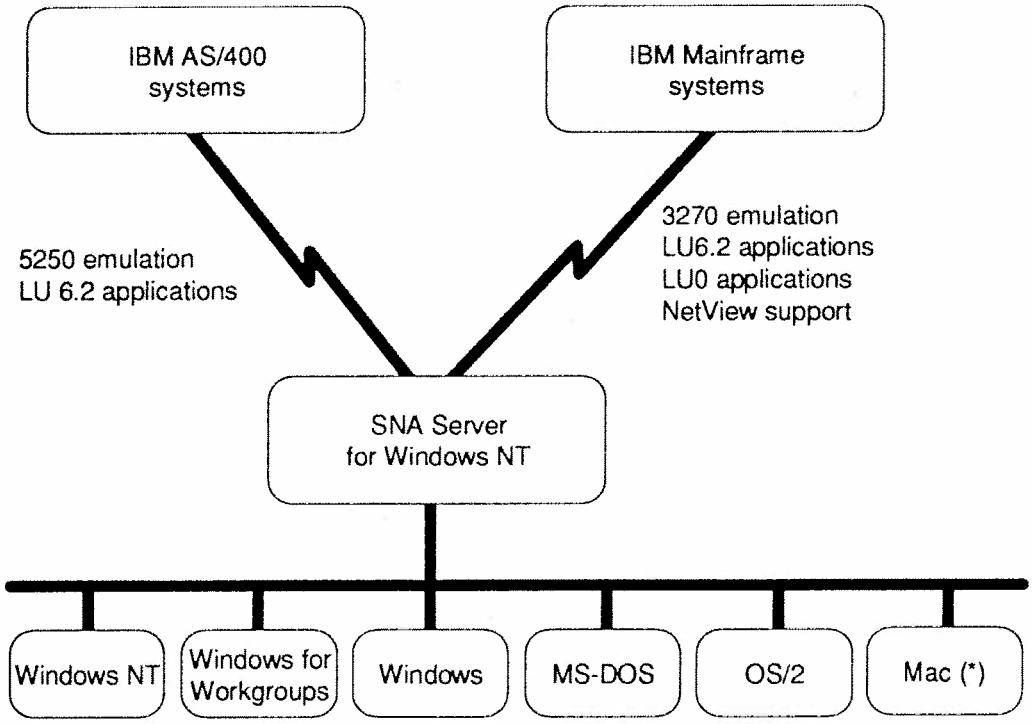
Included in WOSA, Windows SNA API provides a standard interface between the client world (typically) of Microsoft and the server universe of IBM. Both vendors can, of course, function in either client or server mode in accordance with the nature of the application.

To provide developers with the full SNA functionality that is needed, the API specification supports all SNA API categories currently used by programmers. This includes HLLAPI, APPC, CPI-C, LU0, and CSV APIs.

The HLLAPI interface is used with existing 3270– and 5250-based applications. Both the APPC and CPI-C APIs are used to write cooperative applications for the LU 6.2 protocol. The LU0 API is used to gain access to low-level SNA data streams that are often found in banking environments. The CSV API performs character set translations and interfaces with IBM's NetView package.

Microsoft's implementation of Windows SNA API is on a client/server platform. Windows, NT, and DOS clients are supported by a Windows NT-based system performing as an SNA server (see Figure 8.1).

Figure 8.1 SNA Server for Windows NT



(*) Via downstream PU support

SNA Server

The SNA Server for Windows NT offers yet one more option for achieving enterprisewide connectivity. By employing a client/server architecture, this approach improves the flexibility of both host computers and end-user systems. Each client device uses the installed LAN to connect to an appropriate server via standard LAN protocols.

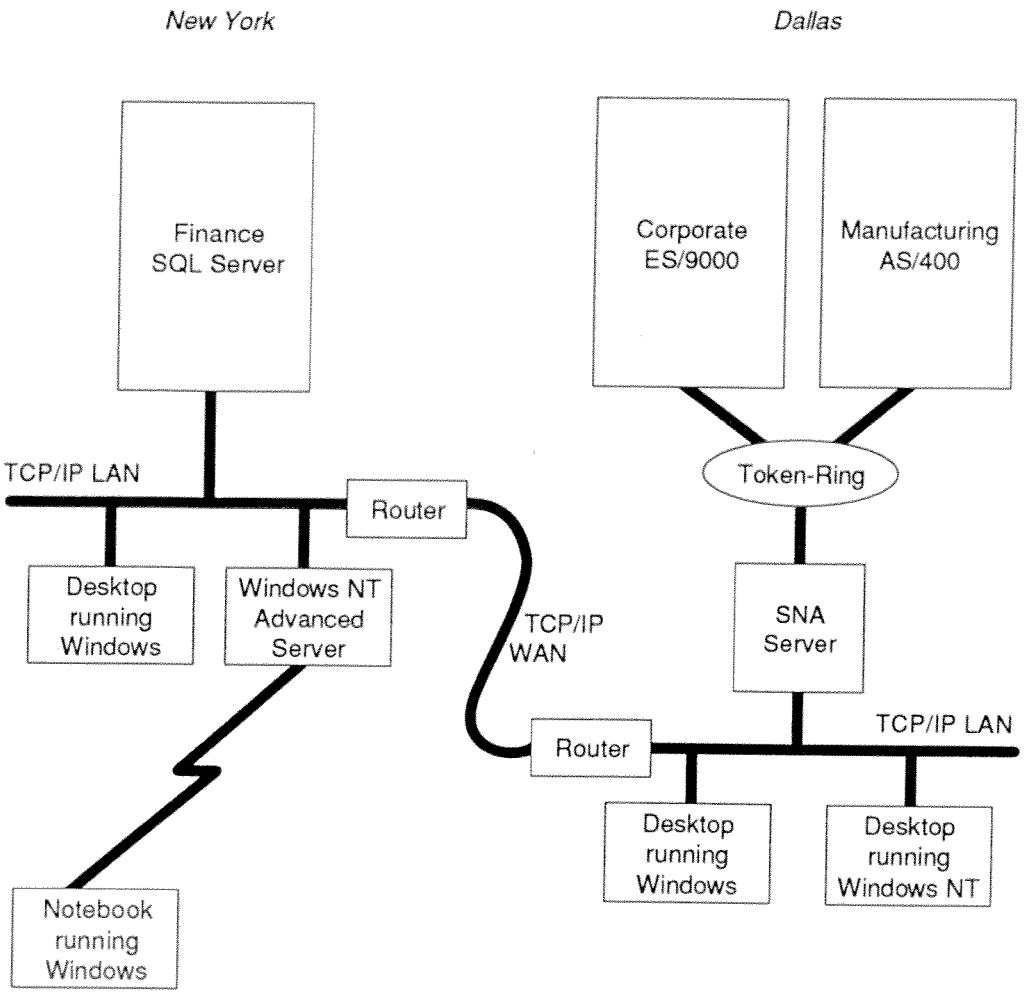
The server then provides shared links to host computers running SNA protocols. Servers also perform much of the processing workload, thus

relieving desktop PCs from many resource-consuming tasks. SNA Server offers numerous tools which facilitate system setup and use, regardless of the platforms and operating systems present in the SNA network.

Some of the key features of SNA Server include:

- it can connect products such as Microsoft Mail and SQL Server to IBM host mail and database systems
- it supports all WOSA SNA APIs for Windows and Windows NT operating systems
- it supports 3270 and 5250 emulators for Windows, NT, DOS, OS/2, and Macintosh operating systems from vendors such as Attachmate, FutureSoft, Eicon Technology, DCA, Wall Data, and IBM
- it supports APPC, CPI-C, CSV, LU0 and HLLAPI APIs for advanced SNA applications in each client environment
- it provides SNA session security and fault tolerance
- it supports client PCs across LAN and wide area network (WAN) bridges, routers, and NT's Remote Access Service (see Figure 8.2)
- it performs automatic load balancing in order to efficiently distribute workload
- it supports a wide variety of LAN protocols (802.2, IPX/SPX, TCP/IP, NetBEUI)
- it uses the C2-level security of NT to safeguard corporate data on the host. C2 is a U.S. government-defined security category which aids in protecting information resources
- it supports NT platforms such as those based on Intel, Alpha, and MIPS processors

Figure 8.2 SNA Server Functions



The capacity of SNA Server is impressive. It supports, for example, up to 500 dependent and 500 independent LU sessions per server. Up to 50 simultaneous host connections (physical units) can be handled per server. It also sustains up to 250 users per server, and can interact with most of the IBM physical unit and logical unit protocols. Examples of the latter are LU0, 1, 2, 3 and LU 6.2.

Comprehensive support for NetView management system is provided by SNA Server. Communication problems are relayed to NetView through

automatic data-link alerts. Bidirectional transmissions to and from NetView are attained via API support. It is also possible to initiate NT commands at the NetView console.

SNA Server device interface driver support is provided for by the adapters shown in Table 8.1. Support for additional adapters is supplied by the adapter manufacturers themselves.

Table 8.1 Supported SNA Communications Adapters

Adapter	Connection
IBM 3278/9 Emulation Adapter	Distributed Function Terminal (DFT)
IBM 3278/9 Advanced Emulation Adapter	DFT
IBM MPCA	Synchronous Data Link Control (SDLC)/X.25
IBM SDLC Adapter	SDLC/X.25
IBM 3270 Connection Model A	DFT
IBM 3270 Connection Model B	DFT
IBM MPA/A	SDLC/X.25
Any NT supported Token Ring or Ethernet Adapter	802.2

Windows NT offers an advantageous platform for PC to host connectivity. SNA Server for Windows NT enables users to implement multiple SNA connectivity products without changing application logic.

Future Trends

Windows SNA API is an important part of the overall WOSA model. SNA remains the most widely installed proprietary network in the world. This standard interface to the SNA environment from Windows platforms enables Microsoft to make further inroads into enterprise computing. At the same time, it strengthens SNA itself by opening it to a wider audience.

Novell, for example, announced plans to add support for the new APIs into its NetWare 3270 LAN Workstation for Windows, a client 3270 emulation program. With standard SNA connectivity available from both the best-selling LAN product and PC operating system platform, IBM's feature-rich network architecture will gain new adherents.

The SNA API, along with the RPC and Sockets APIs, gives WOSA the initial communications support elements it needs to begin achieving its broad connectivity goals. Sockets works with Unix-based applications. RPC enables Windows applications to exchange data with OSF's DCE environment, as well as build distributed solutions. There will undoubtedly be additions to this lineup as WOSA continues to evolve.

Chapter 9

Windows Extensions for Financial Services

XFS Overview

The XFS specification is evolving rapidly despite its relatively short existence. The Banking Systems Vendor Council (BSVC) was officially formed in 1992. By the end of that year, active participants included:

- Andersen Consulting
- DEC
- EDS Corporation
- ICL PLC
- Microsoft
- NCR Corp.
- Olivetti
- Siemens Nixdorf AG
- Tandem Computers
- Unisys

The BSVC released initial specifications in late 1992 for comment by banking industry members and technology suppliers. Review actions are still proceeding. Appendix E identifies some of the primary contacts involved in specification review.

BSVC's mission statement encompasses the following objectives:

- Reduce the cost of software development and maintenance by improving efficiency and productivity of development organizations, reducing the cost of devel-

oper training, and allowing the use of a much larger set of existing applications.

- Improve time-to-market of new applications via simpler development with resulting faster deployment.
- Allow organizations to build modular, scalable systems using a wide array of technology options while leveraging legacy applications.
- Define an architecture workable across a broad range of platforms.
- Encourage utilization of standard interfaces and platforms during application development.
- Reduce user training costs.

In order to implement the objectives stated herein, the BSVC created a corresponding set of strategies. They are as follows:

- Use the Windows operating system in all its variations as their basic platform for client/server computing.
- Use WOSA interfaces and services as the basis for integrating Windows applications into the overall corporate processing strategy.
- Use WOSA components to the maximum extent possible for financial services computing. In all cases, employ existing technology standards whenever feasible.
- Enhance WOSA with XFS to meet the special needs of financial services applications when accessing unique services and devices.

WOSA XFS routines include specifications for access to peripherals peculiar to financial transactions such as passbook/journal/receipt printers, magnetic card readers/writers, and PIN pads. They also support financial transaction messaging and management, network system management, and security procedures. All of these access

capabilities function from Windows client nodes and will be incorporated into the larger WOSA model as appropriate.

The BSVC has identified numerous benefits from utilizing Windows, WOSA, and XFS technologies. They include the following:

- Access financial services and special hardware via the Windows paradigm, thus reducing long-term training costs.
- Leverage the vast body of applications and development tools in the Windows environment.
- Develop applications that will operate on the wide array of Windows platforms and operating systems, from 3.x to NT to Windows for Workgroups.
- Deploy modular, scalable financial services systems that are adaptable to evolutionary changes in the industry.

The core elements of XFS are definitions for a set of APIs, along with their corresponding SPIs, plus supporting services, that allow Windows applications to access financial services software and hardware. These standard APIs allow applications to access differing service providers without altering application code, much like all WOSA components.

Due to the unique nature of devices employed in the financial services industry, initial BSVC focus has been on providing access to them, rather than overall API development. The devices in question have complex interfaces. Standardizing those interfaces will provide immediate productivity gains in financial services computing.

The XFS architecture is shown in Figure 1.8. As shown, applications communicate with services and devices via the XFS Manager, using the API set. Most of these APIs can be initiated synchronously or asynchronously. In the former, the Manager causes the application to wait until the API's function is completed. With asynchronous, the application regains control immediately, while the function is performed in parallel.

The XFS Manager maps specified APIs to the corresponding SPI, which then routes this request to the appropriate service provider. As illustrated in Figure 1.8, the Manager refers to a registration database to direct the API to the proper service provider access point.

It is planned that manufacturers of financial peripheral devices will develop the actual service providers for their units. A setup routine for each device or service will also be implemented in order to define registration database content. The contents of this database will allow an application to seek status information on all available devices and services.

The classes of financial devices shown in Table 9.1 are being implemented for early versions of WOSA XFS. Each vendor is creating APIs for their respective devices. Future additions to this list include devices such as smart cards, card embossers, signature scanners, bar code readers, hologram readers, and other technological innovations.

Table 9.1 Classes of Financial Devices

DEVICE	VENDORS
A. Printers -Receipt and Journal -Passbook and Document	Olivetti, Siemens Nixdorf, NCR ICL, Unisys, DEC
B. Magnetic Stripe Readers/Writers -Swipe -Dip -Motorized -Writeable	ICL, Siemens Nixdorf
C. PIN Pads -With and Without Display -With and Without Encryption	NCR, ICL
D. Cash Dispersers (Note, Coin, Check) -ATMs -Teller Service Centers	NCR, Digital, Siemens Nixdorf
E. Check Readers (MICR and OCR)	Unisys, Olivetti
F. Image Scanners	Unisys, Olivetti

Architectural Issues

Windows and XFS are based on an event-driven, asynchronous model. As described earlier, the XFS design allows an application using its interfaces to operate in both a synchronous and asynchronous manner. Most XFS API functions can be requested in either a synchronous or asynchronous mode. Further details on these modes are as follows:

- Synchronous – The function does not return to the caller until the operation has completed. It is used when an operation can take an indeterminate amount of time to complete, yet the caller wishes to handle the function in a sequential manner. Functions issued in a synchronous manner are referred to as blocking. If a blocking operation cannot be completed immediately, the XFS Manager executes a Windows message loop on behalf of the calling application, thereby keeping the Windows system running. An application can have only one blocking call outstanding at any one time.

- Asynchronous – An asynchronous function is also used for operations which may take an indefinite amount of time to complete. However, the XFS Manager returns to the caller immediately, with an indication that the request has been initiated and is being processed. Upon completion of the request, a Windows notification message is posted to the application window specified in the call, communicating the result of the request.

  When an asynchronous request is issued, the XFS Manager assigns a sequence number, the Request ID, to the request. When the request finishes, the XFS Manager posts a specific message for the request to the application window, specifying the Request ID, and passing a structure containing data on the processed request. Performing an operation in an asynchronous manner, as opposed to a synchronous manner, allows

the application to operate in a Windows' event-driven, message-based manner.

- Immediate – This mode is used for those API functions that are not either synchronous or asynchronous. Typically, immediate APIs are those which do not communicate with a physical device or a service and, hence, are guaranteed to complete immediately, whether successful or not.

  For both synchronous and asynchronous functions, a time-out value can also be specified, indicating the maximum number of milliseconds the application wants to wait for completion of the operation. An application can also implement a request that will wait until completion, no matter how long the request will take, by specifying a value for the time-out.

  Error handling with XFS depends on the mode of operation. Synchronous or immediate mode functions return a value indicating whether the process has executed successfully or not. The returned error code can be requested by a special function call. Asynchronous mode functions behave slightly differently due to the multiplicity of operations possible in this mode.

API Functions

Functions defined by the XFS API are divided into three categories. They are:

- Basic functions (see Table 9.2) that are applicable to all financial services classes.

- Administration functions that are used specifically for the purpose of administering services such as device initialization, reset, etc.

- Specific commands which are used to request device or service-specific functions, and are sent to devices and services as a parameter of the Execute basic function.

Table 9.2 Basic Functions

FUNCTION	DESCRIPTION
1. WFSAcknowledge	Notify WOSA/XFS Manager that dynamic data can be freed
2. WFSCancelAsyncRequest	Cancel an asynchronous activity performed on a specified service provider
3. WFSCancelBlockingCall	Cancel any outstanding blocking operation
4. WFSAsyncExecute	The asynchronous version of WFSExecute
5. WFSAsyncGetStatus	The asynchronous version of WFSGetStatus
6. WFSAsyncLock	The asynchronous version of WFSLock
7. WFSAsyncOpen	The asynchronous version of WFSOpen
8. WFSCleanUp	Terminate the use of the WOSA/XFS Manager
9. WFSClose	Terminate a series of service requests
10. WFSDeregister	Disable event monitoring
11. WFSExecute	Send service-specific commands to the service provider
12. WFSGetInfo	Retrieve information on a service
13. WFSGetLastError	Retrieve information on the last error occurred
14. WFSGetStatus	Retrieve information on the status of a service provider
15. WFSLock	Establish exclusive control over a service provider
16. WFSOpen	Open a session between a service provider and an application
17. WFSRegister	Enable event monitoring
18. WFSStartUp	Start the WOSA/XFS Manager
19. WFSUnlock	Release exclusive control over a service provider
20. WFSIsBlocking	Determine if a blocking call is in process
21. WFSSetBlockingHook	Install an application-specific blocking routine
22. WFSUnhookBlockingHook	Restore the default blocking routine

The SPI is constructed in a manner similar to the API. Some commands are handled exclusively by XFS Manager, and thus are not part of the SPI. There are also some minor variances in the parameters passed at the two interface levels.

Administration Functions and Specific Commands

One segment of the XFS API set deals with administration issues. They are not used by financial applications directly, but perform support services such as initializing a device, and suspending operations. Typical functions are WFSInit, WFSReset, WFSSuspend, etc. Work continues on this aspect of the XFS architecture.

Specific commands apply to a subset of service provider classes.

Therefore, they are not included in the basic or administration functions. As with the latter, their precise characteristics are still being defined.

Implementation

A typical command sequence showing the usage of the proposed APIs is as follows. This example illustrates a series of functions used to print a form on a receipt printer device.

- StartUp (connects the application to the XFS Manager, including version negotiation).
- Open (establishes a session between the application and the device).
- Register (specifies the messages that the application should receive from the service provider).
- Lock (obtains exclusive access to the device from the application).
- Multiple Execute functions, passing a series of specific commands: Select_Form (defines the data structure to be used for the printing).

Chapter 10

Windows Remote Procedure Call API

RPC Overview

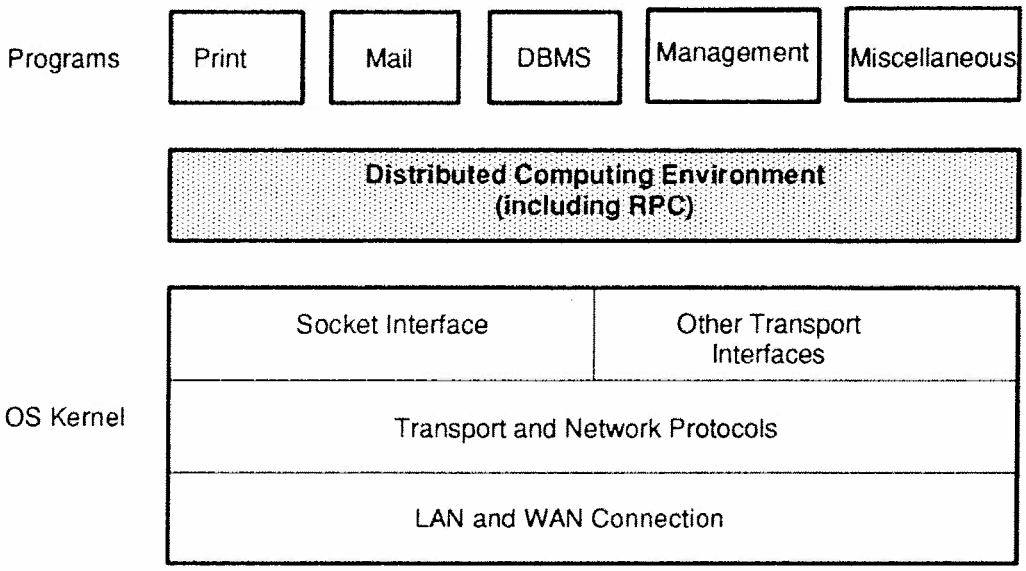
The primary value of RPCs is to simplify the writing of distributed applications. To the user, an RPC masks the complexity of the network, making it appear that a local procedure is being called. RPC functionality is an important capability in a client/server networking environment.

It is widely accepted that all technology users involved with networks desire the following: dedicated processor, shared database, and transparent communications. The preceding may not, in real terms, be available to the user, but must at least appear to be so. The marriage of distributed processing and an RPC facility brings this illusion closer to reality.

A distributed computing configuration gives an assemblage of casually connected systems the appearance of being a single entity. By invoking RPC mechanisms in this distributed environment, the user can look upon the entire aggregate of networked resources as one single system. Programming distributed solutions thus becomes easier to implement.

The relationship of RPC to applications and the networks supporting them is shown in Figure 10.1. Applications such as printing, E-mail, database and network management, and user programs invoke services provided at the distributed computing environment layer. This includes RPC facilities.

Figure 10.1 Relationship of RPC to Applications and Their Networks



The applications remain independent of the underlying transport-level interfaces. Only the DCE itself needs to be aware of the system's transport specifics. The particular transport protocol invoked could be TCP/IP, OSI, or another. The user application simply initiates the RPC mechanism; the rest is transparent to its view.

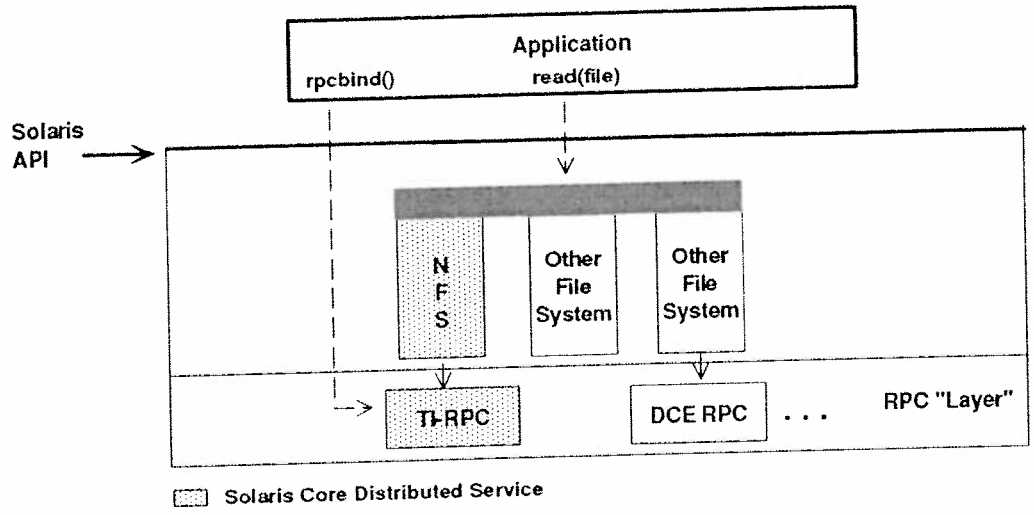
RPCs, as stated, are particularly appropriate for client/server configurations. In this setting, clients request services from network services. For example, a database server can provide requested data to a variety of users and applications by processing RPCs issued by those client entities.

Client/Server Model

RPCs provide two major components that aid in developing distributed applications. There is a language/compiler combination that processes the user's RPC parameters so as to mask complexities associated with remote access. There is also a runtime facility that implements the actual calling mechanisms represented by an RPC. The latter offers transparency relative to underlying transport protocols and architectures as far as application procedures are concerned.

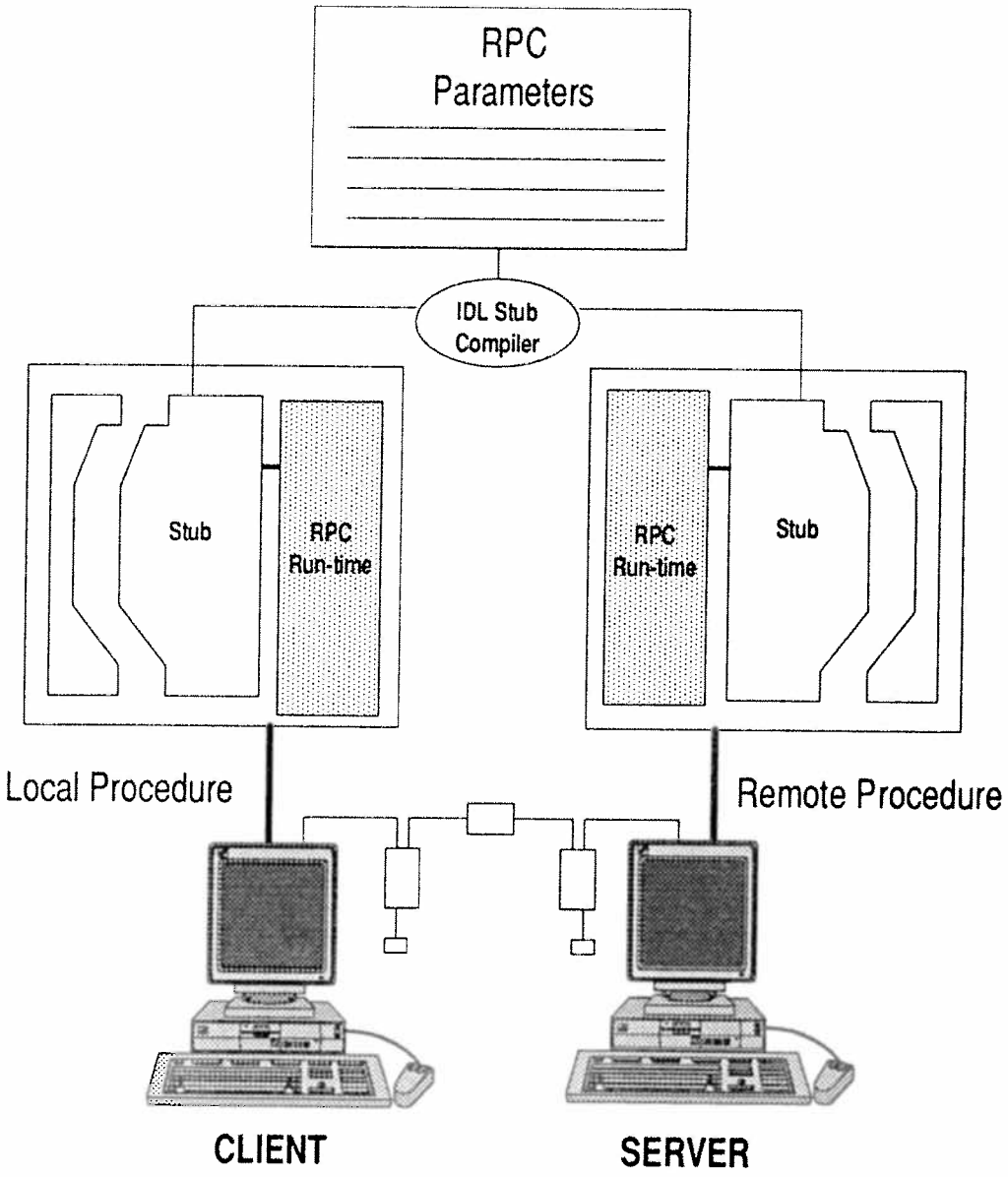
There are many varieties of RPCs in the industry today. Solaris from Sun supports a transport-independent RPC mechanism which is compatible with a widely installed ONC RPC. In addition, Solaris can accommodate a variety of RPC mechanisms, such as that of OSF DCE or the emerging OSI RPC standard, that can coexist peacefully with each other. This allows integrated support for multivendor distributed services, built using these different RPC technologies. It also gives developers more freedom in designing and implementing RPC-based applications that can interoperate in heterogeneous distributed computing environments. (See Figure 10.2).

Figure 10.2 Transport-Independent RPC



The Interface Definition Language (IDL) in OSF's RPC facility is similar to the syntax of the C language. It processes RPC parameters and provides additional constructs appropriate for a network environment (see Figure 10.3). A compiler then translates IDL data into "stubs." To the local procedure, a stub represents the server process it is trying to call. Similarly, on the server side, the remote procedure believes its stub is the client it is trying to service. Thus, stubs are themselves servers to their respective "clients."

Figure 10.3 RPC Operational Sequence



A stub does for the programmer what would otherwise have to be done manually: copying arguments to and from RPC packets, converting data formats, if required, and invoking the RPC runtime facility.

Microsoft's implementation of RPC is compatible with OSF's version. Client procedures using Microsoft's RPC Version 1.x and beyond will

interoperate with any DCE RPC server whose runtime libraries utilize the connection-oriented, virtual model and run over a supported network protocol (TCP/IP, etc.).

There are differences, however, between OSF's and Microsoft's RPC implementations. Among them are:

- Microsoft IDL (MIDL) allows single line comments. OSF IDL leaves this to the C preprocessor.
- MIDL supports more than one union nested in a struct, whereas OSF IDL allows only one.
- MIDL context handles can be declared as pointers to any user defined type, whereas OSF IDL context handles must be of the type "void."
- MIDL allows an attribute to occur as both a type attribute and usage attribute if they are consistent. OSF IDL does not support this duality.
- OSF IDL supports the pipes interprocess mechanism. MIDL does not support pipes.
- MIDL supports constant integer expressions. OSF IDL does not.

These are but a portion of the differences between OSF and Microsoft's RPC approaches. Despite their seemingly wide divergence, however, there are more features that are in harmony than not. Thus, interoperability between them is routinely achieved.

Reviewing an RPC operational sequence in more detail, Figure 10.3 shows that stubs are compiled and linked with the application. When an RPC is initiated, the client application calls a local stub procedure. The client stub code then:

- Retrieves the required RPC parameters from the client application.
- Translates parameters into a standard Network Data Representation (NDR) format in preparation for network transmission.
- Calls client runtime library API functions in order to send RPC parameters to the server.
- Calls server runtime library API functions to accept the incoming data and calls the server stub procedure.
- Server stub procedure converts the NDR-formatted data to native server format.
- The server stub then calls the server procedure.
- The remote procedure executes, perhaps generating return data for the client application.

The process is then reversed. Data is formatted for network transmission, etc. At the completion of this entire procedure, the client application is mostly unaware of the preceding steps. To the application, it appears as if this was a call to a local procedure. No knowledge of low-level network protocols is required. Thus, powerful distributed applications can be developed more easily.

RPC Development Tasks

Due to the remote nature of an RPC process, there are unique steps that must be performed. For example, an interface definition must be created that specifies interface identification, data types, and function prototypes for the remote procedure. An application configuration file will also be needed.

The actual interface definition will be compiled, in the Microsoft arena, using the MIDL compiler. The latter then generates C language stub

and auxiliary files, plus a header file for the client and server. This header file is appended to client and server applications.

A server program is created that can both signal the client as to its availability, as well as monitoring subsequent client requests. A server termination routine must also be included.

Perhaps the most important step associated with the RPC process is functional linking. Both the client and server entities must be linked with their respective stubs, auxiliary files, and runtime libraries.

Interface Definition Language

The IDL, specifically Microsoft's version, shown in Figure 10.3 provides the features needed to extend the C programming language in order to support remote procedure calls. MIDL is not just a variant of C. Rather, it is a formal language that handles control of data transmission over a network. Due to its similarity with C, however, developers are able to quickly learn its characteristics. MIDL compiler options are listed in Table 10.1.

Three prominent features of MIDL are strong data typing, directional attributes, and data transmission support. It is strongly typed because it provides keywords that define all base data type sizes, regardless of the computer on which they are implemented. Specific types as short, small, long, and float replace weakly-typed variables allowed in C.

Directional attributes indicate whether data is transmitted from client to server, server to client, or in both directions. Data transmission over the network is addressed by a transmit function which supports conversion from one data type to another.

Table 10.1 Sample of MIDL Compiler Options

MODE	/mode ms_ext /mode app_config /mode implicit_local	Microsoft extensions mode Application configuration mode Assume non-remote data is [local]
INPUT	/acf filename /I directory /import ms_ext /import ms_nt /import osf /no_def_idir	Specify the attribute configuration file Specify directory for import and include files Compile only needed portions of imported IDL files Compile only needed portions of imported IDL files Assume imported IDL files are compiled separately Ignore the curent and the INCLUDE directories
OUTPUT FILE GENERATION	/client all /client aux /client none /client stub /out directory /server all /server aux /server none /server stub /syntax_check /Zs	Generate client stub and aux files Generate client auxiliary file only Generate no client files Generate client stub file only Destination directory for output files Generate server stub and aux files Generate server aux file only Generate no server files Generate server stub file only Check syntax only; do not generate output files Check syntax only; do not generate output files
OUTPUT FILE NAMES	/caux filename /cstub filename /cswtch filename /Fs filename /header filename /list filename /saux filename /sstub filename	Specify client auxiliary file name Specify client stub file name Specify switch stub file name Generate a listing file Specify header file name Generate a listing file Specify server auxiliary file name Specify server stub file name
C COMPILER AND PREPROCESSOR OPTIONS	/cc_cmd cmd_line /cc_opt options /cpp_cmd cmd_line /cpp_opt options /D name[=def] /no_cpp /U name	Specify C compiler used for stub, aux files Specify options associated with C compiler Specify name of C preprocessor Specify additional C preprocessor options Pass #define name, optional value to C preprocessor Turn off the C preprocessing option Remove any previous definition (undefine)
ENVIRONMENT	/char signed /char unsigned /char ansi7 /env dos /env win 16 /env win32	C compiler default char is signed C compiler default char is unsigned Char values limited to 0-127 MS-DOS client Microsoft Windows 16-bit (WIN 3.x) Microsoft Windows 32-bit (NT)

Building RPC Applications

The specific procedures involved in creating a distributed application can occupy several volumes. Fortunately, these procedures are very similar for all operating systems and platforms. An overview of these steps follows:

- Develop C language and IDL files.
- Generate C language stub files by compiling the IDL and related files via the MIDL compiler.
- Compile C language and stub files via the C compiler.
- Link resulting object files with import libraries for the resident platform.
- Verify that the RPC DLL is in the path.
- Run the client and server distributed applications.

There is obviously much planning and analysis that must accompany the preceding steps, particularly when viewing distributed applications from the perspective of an integrated system. Software tools to aid this process are slowly becoming available in the marketplace.

Chapter 11

Windows Environment Trade-Offs

Background

In the world as viewed by Microsoft, Windows is the center of the universe. Whether 3.x, NT, or later variations– Windows is Microsoft's strategic product. WOSA is an architecture that enhances Windows' viability in the marketplace. Thus, when evaluating WOSA, one must first analyze its Windows foundations.

The Windows operating system has achieved almost de facto standard status as the platform of choice for client applications. As such, it seems to have bred a certain detachment in its developer and sponsor – Microsoft. There appears to be a "we'll go our own way" attitude prevalent among the Microsoft employees, a view that the world needs them more than the other way around.

This is always a dangerous attitude for any corporation. No matter what one's current success, the seeds of destruction lie ever-present in nearby soil, waiting for a proper impetus to initiate the germination cycle. This is not to claim that Microsoft faces imminent dissolution. They have been quick enough on their feet to this point to adapt to market realities. Their disdain, however, for mainstream technology standards, the apparent attitude that they will establish their own standards, may be a sign of less tranquil times ahead.

A case in point is Microsoft's OLE technology. The company has had great success deploying this document and data sharing procedure on standalone machines. Not only is it widely implemented in both the

Windows and Apple Macintosh environments, but users aligned with other systems are also clamoring for an OLE-like capability.

In recognition of its popularity, and in anticipation of the growing need for a similar technology in networked, distributed configurations, Microsoft is working on extending OLE techniques to a new distributed, object-based operating system code-named Cairo. This new product could enable Microsoft to control a vital technology: linking objects across applications in a distributed architecture. It is scheduled for full deployment during the middle years of this decade.

It is important to note, however, that in traditional Microsoft fashion, they are ignoring existing object standards. The Object Management Group (OMG) has been working on a Common Object Request Broker Architecture (CORBA) standard in recent years. It has achieved widespread commitment from major industry players that they will follow its provisions.

Microsoft has not yet committed Cairo to the CORBA standard. They are thus far implementing proprietary procedures. Future interoperability with non-Microsoft environments is therefore uncertain. Special gateways or other techniques may be needed to interoperate outside Microsoft platforms.

Microsoft's explanation for avoiding CORBA may be that the latter is still defining many procedures, such as linking, that are required by the Cairo operating system. OMG continues to develop these capabilities, however, and its CORBA 2.x version will provide these object services in the near future.

In contrast to Microsoft's proprietary approach, Unix International (UI) is developing its own Application Linking technology for Unix platforms that will follow industry standards. They are basing their products on OMB's CORBA which provides them with a foundation for future interoperability. UI represents a large consortium of vendors associated with the Unix marketplace.

Another important vendor grouping is also developing a distributed, OLE-like technology. Apple, IBM, and Novell are working on a system called Exemplar. As with UI, this triumvirate has pledged to follow the CORBA standard. Both of these standards-compliant efforts will provide users with a clear migration path to implement OMG's object linking service once it is fully released.

The bottom line is that Microsoft's competitors are determined that Cairo will not set the tone for distributed object computing technology, which is critical for supporting advanced applications. IBM, UI, et al intend to follow mainstream standards while adding their own value. Microsoft plans to ignore such standards, unless those standards overwhelm the marketplace. Time will determine if Microsoft is quick enough to make whatever shift may become necessary.

Windows 3.x versus OS/2 2.x

Standards compliance, or lack thereof, is not the only soft spot in Microsoft's business plan. Windows 3.x itself, although an extremely popular product among users, has weaknesses in some areas. In comparison with IBM's OS/2 2.x, for example, Windows suffers by comparison. At first glance, both operating systems appear somewhat similar. They each offer a GUI, provide concurrent execution of multiple applications, and enable applications to exploit larger memory resources.

OS/2 2.x goes beyond Windows 3.x in several areas, however. It gives an enhanced user interface that is based on the concept of objects. Known as the Workplace Shell, this feature represents an important step forward in GUI desktop functionality. Also in the object mode category, so-called Extended Attributes provide file system support for object-oriented features.

Table 11.1 illustrates additional differentiating characteristics between Windows and OS/2 2.x. Although features shown in Table 11.1 represent only a sampling of differences, it can be seen that OS/2 2.x is more suitable for a development environment than is Windows. At the same time, OS/2 2.x can do everything that Windows can, and can therefore fulfill all the roles typically associated with Windows platforms.

Table 11.1 Windows 3.x and OS/2 2.x Technology Features

Feature	Windows 3.x	OS/2 2.x
Physical Memory Limit	16 MB+	16 MB+
Virtual Memory Limit	4 times physical	512 MB (disk space)
Multitasking (DOS)	Time slice	Preemptive time slice
Multitasking (Windows or Presentation Manager)	Cooperative	Preemptive time slice
Priority	Static	Dynamic
System Services	Serial	Parallel
Protection Between Applications	Unprotected	Protective
File System	File Allocation Table (FAT)	Enhanced FAT and Performance
Service Support	Error Logging	Error Logging Trace Utilities

Additional Limitations

Additional limitations in the Windows environment include the following:

- Unprotected Resources – Lack of native Windows data protection tools can hamper operations. Users must exit to DOS when attempting to deal with common hard disk problems, such as lost data clusters or damaged File Allocation Tables and directories. These problems can lead to system failure if not rectified.

- Inefficient Use of Resources – Hard disk fragmentation by DOS diminishes system performance. This occurs from DOS' random system of storing data. Eventually, data clusters must be rearranged into a more logical order to improve performance, which can suffer up to a 30% decline over time.

- Program Manager Lacks Flexibility – Windows arbitrarily separates applications and files, making it difficult for users to manage projects and organize their workspace. Although Windows allows users to run multiple applications, it does not provide a convenient method to associate and quickly find the applications and files required for a specific project.

- File Management Lacks Features – Ordinary tasks such as copying, moving, and deleting demand numerous keyboard entries and mouse clicks. File compression and decompression also need simpler user interfaces.

Future

Despite the negatives outlined in preceding paragraphs, there will be no curtains for these Windows. WOSA, and the operating systems upon which it is based, will prosper well into the next millennium. There is simply too much momentum attached to the Windows phenomenon for it to recede within the next ten years. WOSA provides the enhancements needed to ensure future success.

As described earlier, the only cloud on Microsoft's horizon is its proclivity to ignore mainstream technology standards. They have forged their own standards up to now. Whether a volatile technological marketplace will continue to abide by this approach remains to be seen.

In addition, powerful and talented competitors continue to define their own directions. The Unix consortiums, IBM and its multiple alliances, DEC and its technology partners, the world of Sun Microsystems, etc., will challenge each move of Microsoft, and initiatives such as WOSA, every step of the way. As the saying goes: "We are in for interesting times."

Appendices

Appendix A

ODBC Function Summary

Task	Function Name	Conformance	Purpose
Connecting to a Data Source	SQLAllocEnv	Core	Obtains an environment handle. One environment handle is used for one or more connections.
	SQLAllocConnect	Core	Obtains a connection handle.
	SQLConnect	Core	Connects to a specific driver by data source name, user ID, and password.
	SQLDriverConnect	Level 1	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialogs for the user.
	SQLBrowseConnect	Level 2	Returns successive levels of connection attributes and valid attribute values. When a value has been specified for each connection attribute, connects to the data source.
Obtaining Information about a Driver and Data Source	SQLDataSources	Level 2	Returns the list of available data sources.
	SQLGetInfo	Level 1	Returns information about a specific driver and data source.
	SQLGetFunctions	Level 1	Returns supported driver functions.
	SQLGetTypeInfo	Level 1	Returns information about supported data types.
Setting and Retrieving Driver Options	SQLSetConnectOption	Level 1	Sets a connection option.
	SQLGetConnectOption	Level 1	Returns the value of a connection option.

Task	Function Name	Conformance	Purpose
Setting and Retrieving Driver Options *(Continued)*	SQLSetStmtOption	Level 1	Sets a statement option.
	SQLGetStmtOption	Level 1	Returns the value of a statement option.
Preparing SQL Requests	SQLAllocStmt	Core	Allocates a statement handle.
	SQLPrepare	Core	Prepares an SQL statement for later execution.
	SQLSetParam	Core	Assigns storage for a parameter in an SQL statement.
	SQLParamOptions	Level 2	Specifies the use of multiple values for parameters.
	SQLGetCursorName	Core	Returns the cursor name associated with a statement handle.
	SQLSetCursorName	Core	Specifies a cursor name.
	SQLSetScrollOptions	Level 2	Sets options that control cursor behavior.
Submitting Requests	SQLExecute	Core	Executes a prepared statement.
	SQLExecDirect	Core	Executes a statement.
	SQLNativeSql	Level 2	Returns the text of an SQL statement as translated by the driver.
	SQLDescribeParam	Level 2	Returns the description for a specific parameter in a statement.
	SQLNumParams	Level 2	Returns the number of parameters in a statement.
	SQLParamData	Level 1	Used in conjunction with **SQLPutData** to supply parameter data at execution time. (Useful for long data values.)
	SQLPutData	Level 1	Send part or all of a data value for a parameter. (Useful for long data values.)
Retrieving Results and Information about Results	SQLRowCount	Core	Returns the number of rows affected by an insert, update, or delete request.

Task	Function Name	Conformance	Purpose
Retrieving Results and Information about Results *(Continued)*	SQLNumResultCols	Core	Returns the number of columns in the result set.
	SQLDescribeCol	Core	Describes a column in the result set.
	SQLColAttributes	Core	Describes attributes of a column in the result set.
	SQLBindCol	Core	Assigns storage for a result column and specifies the data type.
	SQLFetch	Core	Returns a result row.
	SQLExtendedFetch	Level 2	Returns multiple result rows.
	SQLGetData	Level 1	Returns part or all of one column of one row of a result set. (Useful for long data values.)
	SQLSetPos	Level 2	Positions a cursor within a fetched block of data.
	SQLMoreResults	Level 2	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
	SQLError	Core	Returns additional error or status information.
Obtaining information about the data source's system tables (catalog functions)	SQLColumnPrivileges	Level 2	Returns a list of columns and associated privileges for one or more tables.
	SQLColumns	Level 1	Returns the list of column names in specified tables.
	SQLForeignKeys	Level 2	Returns a list of column names that comprise foreign keys, if they exist for a specified table.
	SQLPrimaryKeys	Level 2	Returns the list of column name(s) that comprise the primary key for a table.

Task	Function Name	Conformance	Purpose
Obtaining information about the data source's system tables (catalog functions) *(Continued)*	SQL Procedure Columnns	Level 2	Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.
	SQLProcedures	Level 2	Returns the list of procedure names stored in a specific data source.
	SQLSpecialColumns	Level 1	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
	SQLStatistics	Level 1	Returns statistics about a single table and the list of indexes associated with the table.
	SQLTablePrivileges	Level 2	Returns a list of tables and the privileges associated with each table.
	SQLTables	Level 1	Returns the list of table names stored in a specific data source.
Terminating a Statement	SQLFreeStmt	Core	Ends statement processing and closes the associated cursor, discards panding results, and, optionally, frees all resources associated with the statement handle.
	SQLCancel	Core	Cancels an SQL statement.
	SQLTransact	Core	Commits or rolls back a transaction.
Terminating a Connection	SQLDisconnect	Core	Closes the connection.
	SQLFreeConnect	Core	Releases the connection handle.
	SQLFreeEnv	Core	Releases the environment handle.

Appendix B

MAPI-related Terms, Associated Concepts, and Alternative Models

1. Common Mail Calls (CMC) – The X.400 API Association (XAPIA) has created a committee to define a standard messaging interface for user applications. In its early stages, members consist of a subset of XAPIA members including IBM and Microsoft itself. This is a good example of vendors "covering all bets," as witnessed by Microsoft's participation while also promoting MAPI.

If CMC "bears fruit," its specification will contend with VIM and Simple MAPI. Microsoft has pledged to migrate from MAPI if CMC is manifested with a substantive solution.

2. Distributed Office Support System (DISOSS) – IBM runs this package under VMS and VSE mainframe operating systems. It handles E-mail and related documents in server mode for a variety of clients. DISOSS uses IBM's SNA Distribution Services (SNADS) to transport mail and documents between systems. It supports Document Content Architecture (DCA) and Document Interchange Architecture (DIA) for controlling complex documents. DISOSS is being slowly supplanted by X.400 services as well as by IBM's own APPC (LU 6.2) peer-to-peer protocols.

3. Electronic Data Interchange (EDI) – Business documents transmitted over networks are increasingly following this international standard. Particularly adaptable to transactions that use structured forms, EDI provides standard formatting and data transport guidelines.

4. Enterprise Messaging Server (EMS) – Will function as a Windows NT mail server. Messaging applications from third-party developers will access EMS via MAPI or XAPIA's X.400 specification. EMS is based on X.400 messaging technology.

5. Internet Protocol (IP) – Many messaging methodologies operate over TCP/IP networks. IP performs routing, as well as mandating record formats of messages traversing such networks.

6. Message Handling Service (MHS) – Used by Novell to provide messaging services in NetWare configurations. It communicates with other messaging mechanisms, such as X.400, through third-party gateways.

7. Multipurpose Internet Mail Extensions (MIME) – Allows a message on the Internet to contain text, binary, or other formatted data. MIME encodes this data into a format compatible with the Internet's own Simple Mail Transfer Protocol (SMTP).

8. NetWare Global Messaging (NGM) – Enables Novell to support four messaging protocols on one NetWare server. Protocols supported are SMTP, IBM's SNADS, Novell's own Standard Message Format used with MHS, and the Unix standard. NGM also functions as a NetWare Loadable Module.

9. NetWare Loadable Module (NLM) – A program running on a NetWare server, it can do stand-alone tasks such as network monitoring. An NLM can also perform as a server in a client/server system where it could be managing E-mail processing.

10. Open Collaborative Environment (OCE) – Apple's response to the demand for workgroup computing includes messaging support, directory services and related APIs. Closely associated with the Macintosh operating system.

11. Open Messaging Interface (OMI) – A precursor to VIM, Lotus offered this API as a vehicle for electronic messaging.

12. Standard Message Format (SMF) – Novell's API for NetWare's MHS, it has been steadily upgraded to support their more recent NGM service.

13. Simple Mail Transfer Protocol – SMTP has achieved standards status in Internet and other Unix environments for support of electronic messaging. The X.400 standard has begun to make inroads into its heretofore widespread support.

14. SNA distribution Services – IBM's message transport protocol for SNA configurations allows supporting E-mail offerings to interact with SNADS-compliant mainframes.

15. Service Provider Interface (SPI) – Microsoft provides this layer between back-end service providers, such as database systems, and Windows applications. This isolates service provider vagaries from the application. SPI is an important element of WOSA.

16. Vendor Independent Messaging – VIM was spearheaded by Lotus to support a cross-platform messaging API. It evolved from the OMI API and is able to access other services such as X.400 and MHS.

17. X.400 – An international standard for global messaging developed by the Consultative Committee for International Telegraph and Telephone (CCITT), it is not specific to any operating system or vendor environment. It is the most widely supported of the Open Systems Interconnect standards.

18. X.500 – This CCITT standard is designed to provide directory services for X.400. Its acceptance in the commercial marketplace will directly impact the future success of X.400 as well.

Appendix C

Windows for Workgroups

A Microsoft product that will play an important role in the future evolution of WOSA is Windows for Workgroups (WFW). It is a networking version of Windows 3.x, and may ultimately even approach the staggering sales figures of Windows itself.

WFW facilitates the introduction of networking to workgroup environments, and simplifies the expansion of existing small networks. It is clearly aimed at the relative novice, containing features which eliminate the need for a technology guru when configuring and operating a comparatively small local area network.

The "Smart Setup" module of WFW enables users of varying expertise levels to install the product and get a network up and running. Smart Setup provides intelligent defaults for networking software so that non-technical users can manage the task of configuring network software.

Naturally, when arbitrary defaults are employed, some degree of operational efficiency may be compromised. It depends on the nature of the network configuration. The bottom line, however, is that casual users can get a network started from scratch, and keep it running for an indefinite period of time.

Organizations with an existing network can implement WFW as a client. This will enable them to add new capabilities such as workgroup-type applications, scheduling, E-mail, and network Dynamic Data Exchange (DDE). The product is compatible with Microsoft LAN Manager and Novell NetWare, and allows users to simultaneously access both at the same time. WFW will also work on Banyan VINES.

Some of the new features offered by WFW include:

- Microsoft Mail – A full-featured E-mail service, Mail allows users to read, compose, forward and reply to E-mail messages, as well as to manage messages they receive. It also allows users to send documents from other business applications – including word processors and spreadsheets, or OLE objects, such as sound annotation — as part of a mail message that can be directly opened and read by the recipient. Microsoft Mail can be plugged into an existing mail system that users may already have, providing an easy-to-use graphical front end. For workgroups that need to connect to other workgroups or access gateways to other E–mail systems, Microsoft is separately providing Mail extensions by which users can connect to other E–mail systems and services.

- Schedule+ – A full-featured graphical scheduling application, Microsoft Schedule+ allows users to schedule group meetings and manage daily calendars and task lists electronically. This minimizes the time it takes to set up group meetings by providing shared access to multiple online calendars.

- Network DDE – This feature allows users to create compound documents that share data across the network. For example, users can create reports by importing spreadsheet charts from other users' machines into their word processing documents. Live links between documents automatically update any information that changes in the original document. These links can be created between any of the DDE-aware applications currently shipping for Windows.

- Security – WFW strives to strike a balance between ease of access and security. Users can make files, data and printers available to others, so that people within the workgroup can reach shared resources with the

click of a button. Users can also restrict access to shared resources as desired, including requiring a password for access. Other users on the system can either be given full access to a directory, or granted permission to read files, but not modify them in any way. Users who want more sophisticated security can store important data on a server and protect access to that data through security features of the server operating system.

- Message Interface – As part of WOSA, WFW supports the Simple MAPI interface in conjunction with Microsoft Mail. Developing mail-enabled applications is aided with Simple MAPI, because developers are free of the complexity of a back-end messaging system. A developer can, for example, write applications to route forms electronically for approval, or to allow users to delegate parts of a document to different individuals and automatically assemble the finished document from the completed parts.

- Improvements to Windows File Manager – WFW includes improvements to Windows File Manager that improves information sharing. A toolbar allows users to move among multiple servers and directories to find scattered corporate data, which simplifies information access. WFW loads Windows-based and Microsoft LAN Manager networking software into protected mode memory, freeing up more MS-DOS memory below 640K for applications, terminate-and-stay-resident programs (TSRs), and other network drivers.

Appendix D

NT Versus the World

Microsoft Windows is the most widely-used PC-based GUI on the market today. DOS, its underlying operating system, is considered inadequate, however, for meeting the challenges of distributed computing. This is not to claim DOS will disappear any time soon.

On the contrary, DOS and Windows are so firmly entrenched that they will prosper well into the next century. There is a recognition by many vendors and users, however, that new solutions are needed. Products such as NT, OS/2, SCO Unix 3.2 and SCO Open Desktop from Santa Cruz Operation, Solaris from SunSoft, and Unix System V, Release 4.2 from Unix System Laboratories and UNIVEL (the latter two being Novell entities) represent new alternatives for the desktop.

They all offer a high degree of application compatibility, interoperability, and network savvy. The relative merits of OS/2 have been discussed in Chapter 11. Here, a comparison between Unix and NT is offered. This information is based on data contained in a report published by Unix International, and should be interpreted in that light.

- Availability – Unix for the desktop has been available for several years from multiple vendors on virtually every system architecture currently available. NT is a more recent product, of course, reflecting Microsoft's vision for enterprisewide computing, at least at the client/server level. NT is targeted for Intel, MIPS, and DEC's Alpha platforms.

- Pricing – Unix and NT are relatively close in initial cost. It should be noted, however, that Microsoft has

traditionally been very aggressive in their pricing strategy. It is possible they may precipitously drop their selling price at some point in the marketing cycle.

- Memory and Disk Requirements – There is no significant difference in this area. In both cases, specific resource needs are determined by characteristics of the overall system being served.

- Functionality – Both Unix and NT offer multitasking, multiprocessor support, and fault tolerant capabilities. NT, however, lacks multiuser support.

- Networking – Desktop Unix supports Sun Microsystem's ubiquitous Network File System. NT will avail itself of this de facto standard via third-party solutions.

- Standards Compliance – Unix supports POSIX and X/Open standards as an integral part of the operating system. NT does not support X/Open's Portability Guide, and only follows POSIX 1003.1 dictates through the use of a POSIX-compliant subsystem. Thus, NT applications themselves are not POSIX-compliant. They may set their own standard, however.

Six major vendors – Hewlett-Packard, IBM, SCO, Sun Microsystems, Univel, and Unix System Laboratories – have banded together, ostensibly to harmonize their various Unix versions, but more to oppose a feared NT onslaught. They have declared their intention to build a common desktop environment and create a new set of APIs to help independent software developers write to the common desktop.

The diversity of interests represented by this group makes for strange bedfellows. They are obviously driven by a fear that NT will effectively replace whatever presence Unix has on the desktop. It is a well-justified fear.

Appendix E

Banking System Vendor Council Contacts

The vendors participating in the development of the WOSA Extensions for Financial Services are listed below, with a contact for each.

Andersen Consulting

Dan Steinman
Chicago, IL
Phone: +1 312 507 7856

Mark J. Allaby
Andersen Consulting AG
Binzmuhlestrasse 14
8050 Zurich, Switzerland
Phone: +41 1 308 1555
Fax: +1 41 1 308 1850

DEC

Robert Waller
Digital Equipment Corp.
200 Forest Street
Marlboro, MA 01752
Phone: +1 508 467 9746

Svante Burström
Retail Banking Group
Digital Equipment Corp. BCFI AB
Box 904
175 29 Jarfalla, Sweden
Phone: +46 8 759 49 58
Fax: +46 8 739 86 76

Pedro Muñoz
Marketing Manager
Retail Banking, Europe
Digital Equipment Co. Limited
Enterprise House
190 High Holborn
London WC1V 7BE, England
Phone: +44 71 831 8282
Fax: +44 71 405 6477

EDS Corporation

Bradley D. Hallin
Division V.P., Chief Technologist
EDS Corporation
LFI Division
5400 Legacy Drive MS: B5-2D-09
Plano, TX 75024

Phone: +1 214 604 4990
Fax: +1 214 604 1172

ICL PLC

Mike Shaw
Manager, Architecture and Integration
ICL Financial Services Business
International House
292 High Street
Slough
Berkshire LS1 1NB, England

Phone: +44 753 555126
Fax: +44 753 555343

Microsoft Corporation

Tom Sherrard
Marketing Manager
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Phone: +1 206 936 4526
Fax: +1 206 936 7329
Internet email: tomsh@microsoft.com
CompuServe ID: 70673,2167

Randi Bussin
Marketing Manager
Microsoft Europe
Tour Pacific
Cedex 77
92977 Paris La Defense - France

Phone: +33 1 46 35 10 01
Fax: +33 1 46 35 10 30
Internet email: randib@microsoft.com

NCR Corporation

C.R. (Bob) Lischer
Director, Engineering
Financial Systems Business Unit
NCR Corporation
Brown and Caldwell Streets
Dayton, OH 45479

Phone: +1 513 445 7775
Fax: +1 513 445 7191

Ing. C. Olivetti & C.S.p.A., and ISC-Bunker Ramo: an Olivetti Company

Michael Ehrenberg
Director
Olivetti Advanced Finance Development
2 Enterprise Drive
Shelton, CT 06484-4636

Phone: +1 203 337 1598
Fax: +1 203 337 1667

Siemens Nixdorf

Ralph Mueller
Section Manager
Application Software
Banking and Insurance Division
Siemens Nixdorf
Informationssysteme AG
Herriotstrasse 7
W-6000 Frankfurt 71, Germany

Phone: +49 69 6682 3554
Fax: +49 69 6682 1021

Petra Hirtz-Bokamper
Product Manager
Finance and Insurance
Division/Banking
Siemens Nixdorf
Informationssysteme AG
Furstenallee 7
W-4790 Paderborn, Germany

Phone: +49 5251 8 11396
Fax: +49 5251 8 11381

Tandem Computers

Chip Greenlee
Manager, Finance Industries
Tandem Computers
19191 Vallco Parkway, LOC 4-26
Cupertino, CA 95014-2525

Phone: +1 408 285 2110
Fax: +1 408 255 8067

Unisys

Andrew Mellor
Program Manager
Unisys Corporation MS B120
Township Line and Jolly Road
Blue Bell, PA 19424

Phone: +1 215 986 3426
Fax: +1 215 986 6791