Novell AppWare

A System for Developing Network Applications

White Paper

July 1993

NOVELL

462-000375-001

© 1993 by Novell, Inc., 122 East 1700 South, Provo, Utah 84606, USA

All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express prior written consent of the publisher.

For more information about Novell products,
contact Novell as follows:
In the U.S. or Canada: Call 1-800-NETWARE (1-800-638-9273).
In all other locations, contact your local Novell office or call 1-801-429-5588.

Novell, the N design, NetWare, Btrieve, Novell DOS are registered trademarks, and AppWare, AppWare Bus, AppWare Foundation, AppWare Loadable Module (ALM), IPX, NetWare Loadable Module, Novell Visual AppBuilder, ODI, and Technical Support Alliance are trademarks of Novell, Inc. UNIX is a registered trademark of Unix System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc.

Macintosh is a registered trademark, and MPW is a trademark of Apple Computer, Inc. Banyan is a registered trademark of Banyan Systems, Inc. Easel is a registered trademark of Easel Corporation. Gupta is a trademark of Gupta Technologies, Inc. Intel is a registered trademark of Intel Corporation. OS/2 is a registered trademark, and SAA is a trademark of International Business Machines Corporation. Excel, LAN Manager and Windows NT are trademarks of Microsoft Corporation. Motorola is a registered trademark of Motorola, Inc. Oracle is a registered trademark of Oracle Corporation. Powersoft is a trademark of Powersoft Corporation. ONC and TIRPC is a trademark of Sun Microsystems, Inc. Sybase is a registered trademark of Sybase, Inc. UnixWare is a registered trademark of Univel, Inc. X/Open is a trademark of X/Open Company, Ltd.

Contents

Using This Document

This paper provides an overview of AppWare™, Novell's system for developing network applications. It describes the components of the AppWare system and explains how developers and users will interact with this new layer of software.

The AppWare system is comprised of two major software components: the AppWare Foundation and the AppWare Bus™. These components provide a consistent set of platform-independent, network-independent and service-focused interfaces and engines that accommodate the needs of commercial and corporate developers who need to create network applications quickly and easily.

This paper is divided into the following sections.

AppWare: An Overview gives an overview of the development challenges AppWare addresses.

AppWare: A Technical Description discusses the two major components of AppWare, as well as the wide range of third-party tools and support services for AppWare.

Appendices cover the technologies discussed in this paper in more detail.

Glossary describes the key terms introduced in this paper.

AppWare: An Overview

What Is AppWare?

AppWare is a new layer of software that leverages today's popular operating systems, development tools and applications. AppWare allows both commercial and corporate software developers to create and deploy network applications more quickly and easily. The role of the AppWare layer is similar to the role of the operating system layer. By shielding developers from the complexities of hardware, the operating system has accelerated the growth of new desktop applications. Similarly, AppWare will shield developers from the complexities of networks and accelerate the growth of network applications. As a result, users can more easily run a wide variety of applications that take full advantage of the network and its powerful services.

The Need for AppWare

AppWare provides a solution to two critical challenges currently facing the software development community:

- It simplifies the development of network applications in a complex, heterogeneous environment.
- It shortens the network application development process and makes it more efficient.

Developers who create network applications must deal with the growing numbers of operating systems, development application programming interfaces (APIs), computing standards and development toolkits available today. AppWare hides these complexities from developers by providing them one uniform set of APIs for accessing different operating systems, graphical user interfaces (GUIs) and network services. Using AppWare, programmers can write the application code once and run the application on different operating systems and networks.

Even with a single API and one code path, however, corporate and vertical software developers cannot afford to create applications using line-by-line coding — a time-consuming task that requires specialized skills. AppWare gives programmers the ability to build applications by using large-grained, interchangeable software components. This way, developers can quickly construct powerful, reliable applications without writing a single line of code.

With AppWare, developers can quickly and easily build applications that provide users the full benefits of the power of their networks. As the leading provider of network operating systems and services, Novell has developed AppWare to provide the underlying APIs, development technologies and services required to successfully write network applications.

Development Challenges

The Complexity of Developing Network Applications in a Heterogeneous Environment

On average, one major new operating system or service API is delivered to developers every 45 days. This rapid pace makes it virtually impossible for application developers to keep up with emerging technologies. Resource constraints force developers to choose among the available alternatives. Often, developers can only afford to focus on one platform or operating system, limiting the markets and minimizing the users' needs that their products can address. Consider this challenge in the following contexts:

Multiple Networking Models

Application developers are currently faced with an unprecedented need for network access and functionality. Now that networks dominate the computing environment, software developers are shifting their focus from writing standalone applications to creating network applications. Developers must contend with a variety of network models to deliver the services and data their users require.

The widespread presence of heterogeneous computing environments has made the choice of development platforms difficult, as developers try to anticipate the best markets for their products. In the near future, network accessibility and functionality will determine the success or failure of many organizations.

Multiple Desktop Platforms

To support multiple platforms, developers must learn all the details of the desktop platforms on which their applications run. For example, if a developer had to write a million lines of C code to create an application for MS Windows, that developer would likely have to rewrite most of that code for each platform on which the application will run, such as Macintosh, UNIX® or OS/2. Furthermore, developers must know how to navigate and control the networks that link these various desktop platforms. Thus, in most cases, the knowledge, skills and resources required to create network applications have been prohibitive.

The Inefficiency of Developing Network Applications

Assume that the complexity problem was solved and programmers had access to all major operating systems and networks through one standard set of APIs. Even in this case, the traditional application development process requires writing code from scratch using third generation languages (3GLs) such as C or C++. Typically, this method of application development is used by horizontal application vendors who work on multi-million dollar projects that can take years to complete.

Corporate and vertical software developers, however, can no longer afford to create applications using this method. Through several corporate advisory councils, Novell has learned that desktop custom applications have about one-fifth the lifespan and take 50 percent more time to create than mainframe custom applications. Corporations cannot downsize their mainframe applications without a more efficient method of developing and deploying applications. As a result, corporations are running into severe roadblocks when they attempt to replicate mission-critical mainframe applications on desktops and networks.

The industry has reached a point where developers are demanding a more effective response to these problems than today's tools and techniques can provide. Just as the operating system solved the application crisis for desktop computers, AppWare solves the current crisis for network applications (see Figure 1).

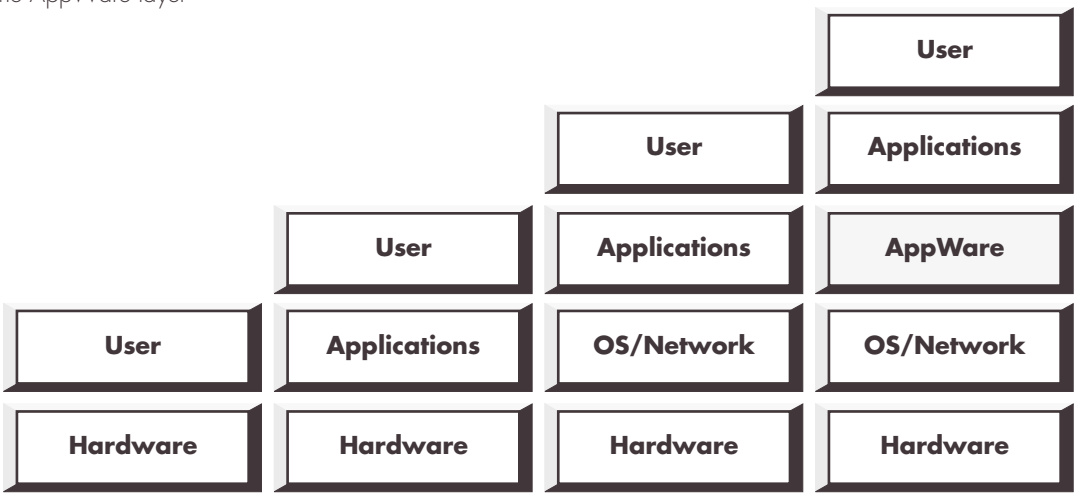
The AppWare Solution

AppWare is not an operating system or application. AppWare is a new layer of software that provides two components of technology to solve the two primary challenges described previously. These two components are the AppWare Foundation™ and the AppWare Bus.

The AppWare Foundation is a "fire wall" that insulates programmers from the growing complexities of multiple operating systems, GUIs and networks. The AppWare Foundation provides 3GL application programmers with a consistent, cross-platform set of APIs. By writing to this single API set, developers can access existing GUIs, operating systems and network services.

The second AppWare component, the AppWare Bus, provides large-grained, interchangeable software modules that corporate and vertical developers can use and reuse to quickly construct new network applications, without having to write code. These software components are called AppWare Loadable Modules™ (ALMs).

Figure 1. The AppWare layer



The AppWare Bus and ALMs are the software equivalent of the PC mother board and interchangeable plug-in cords. When ALMs are plugged into the AppWare Bus, their functionality becomes rapidly accessible for building new applications. In addition, third parties can offer their technology in the form of new ALMs.

The AppWare solution provides the following benefits:

- AppWare enables developers to easily access network services from a variety of desktop platforms. AppWare makes it as easy to incorporate messaging, telephony, multimedia, imaging and other network capabilities into an application as it is to incorporate basic file and print services.
- Traditional programmers can use one set of APIs, instead of having to master the underlying details of how to define, access and use the wide array of services available today. As a result, developers can concentrate on application-specific functionality.
- Reusing existing software modules greatly reduces application development time. AppWare enables developers to construct applications with only a fraction of the time and effort required when building an infrastructure along with an application.
- Once familiar with AppWare, developers can reuse its components as needed, rather than having to recreate the same components for each new application.
- AppWare is open at all levels and will be available on all major operating systems and networks. Therefore, it opens broader application markets than have previously existed.
- Because AppWare is inherently multiplatform in its design and capabilities, it relieves developers of most of the effort required to build and maintain different application versions for each platform they support. Applications written on top of the AppWare Foundation API set can be recompiled to run on DOS, MS Windows, Windows/NT, OS/2, UnixWare and the Macintosh desktops, as well as NetWare® and UnixWare servers.

Novell intends to work with developers, development tool vendors, hardware and operating system suppliers, and other third parties to make AppWare a standard for network application development. AppWare will remain open and extendible and will incorporate important development and interface standards. Therefore, AppWare can be welcomed into companies that must protect themselves from the vagaries of proprietary system components.

AppWare: A Technical Description

This section starts by describing the heterogenous nature of today's business environment which has led to the developmnet of AppWare. This description is followed by a representation of the AppWare components and the third-party tools that support it. The section concludes with an example of using AppWare for developing network appplications.

The Business Computing Environment

Contemporary computing environments consist of multiple hardware and operating system platforms. These platforms range from network or application servers, such as NetWare or UNIX, to mainframe systems that process transactions and act as repositories for consolidated enterprise information.

New facets of these environments include mobile computing and dedicated systems. Mobile computing encompasses notebook systems and personal digital devices, all of which need access to network services and messaging support. Dedicated systems consist of microprocessors or computing systems embedded into a broad range of devices, from machines on the shop floor to hand-held data acquisition devices for inventory management in retail settings.

As microprocessor technology continues to decrease in cost, computing intelligence will find a role in almost every piece of machinery and in every appliance imaginable. Examples range from lathes on the shop floor to hand-held devices. The term ubiquitous computing applies to any intelligent device that incorporates some computer technology. To reap the maximum benefits from these intelligent devices, developers must include them in the networked environment so they can access, as well as provide, network services.

As Figure 2 shows, a heterogeneous computing environment clearly exists and includes the following:

- At the desktop, MS Windows, Windows/NT, Macintosh, OS/2, Novell DOS® and UnixWare Personal Edition provide operating system services.
- At the server, NetWare and UnixWare provide network services that cover everything from basic file and print to database and telephony services.
- Heterogeneous networking environments include IBM's SAA, USL/SunSoft's Open Network Computing (ONC), the Open Software Foundation's (OSF) Distributed Computing Environment (DCE), as well as Novell's NetWare operating system.
- Mainframe and mini-computer platforms host legacy systems, high-volume transaction systems and an enterprise's repositories of consolidated information.
- Intelligent (or ubiquitous computing) devices such as manufacturing equipment on the shop floor can be integrated through the network with a CAD engineer's design workstation, so that they may be reconfigured electronically.

Figure 2. The heterogeneous computing environment

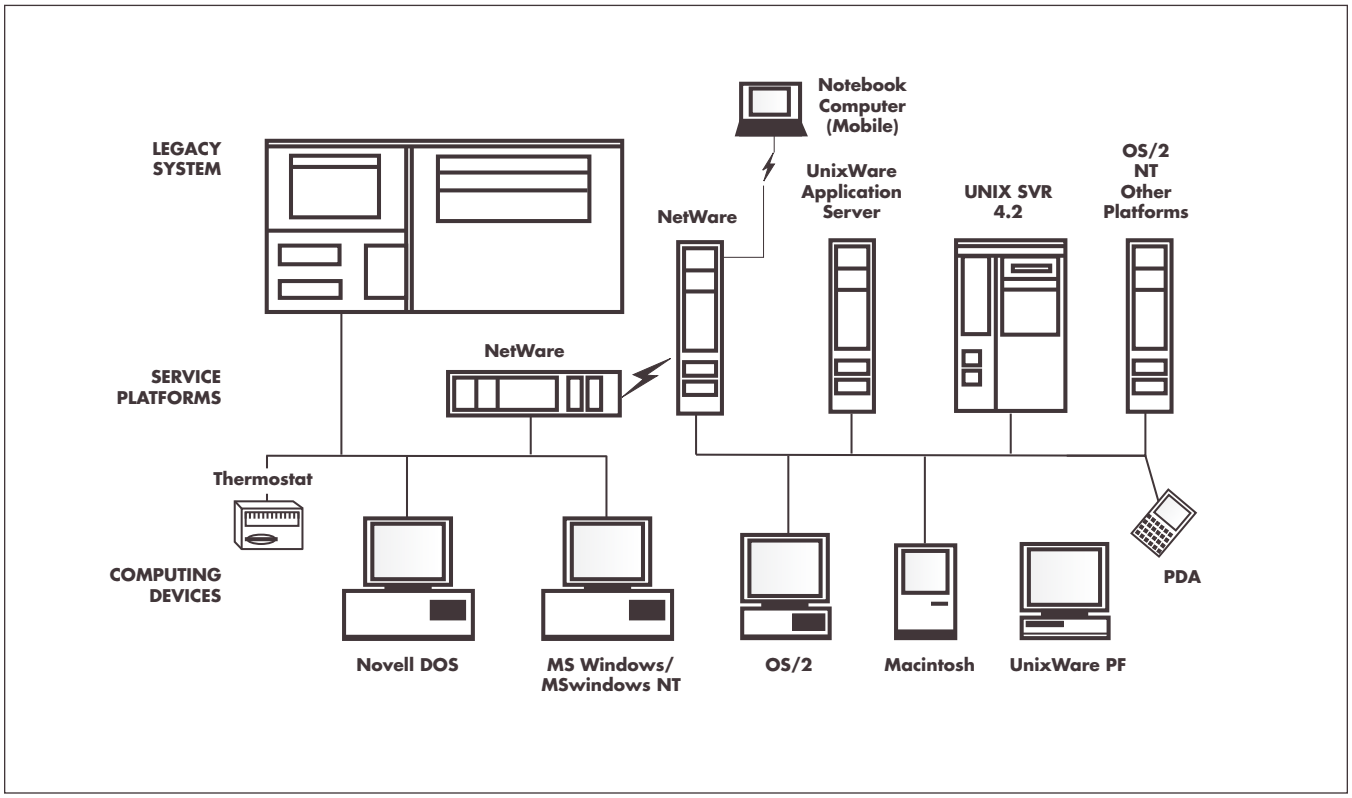
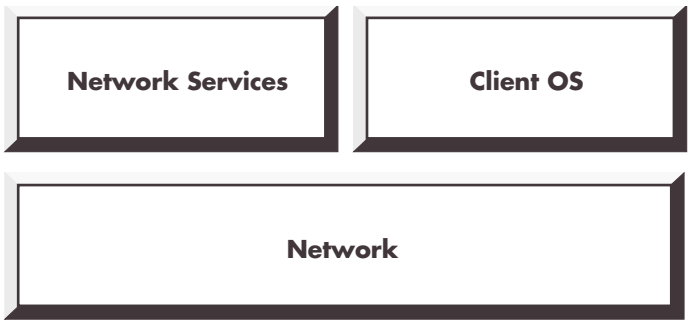


Figure 3 shows the three categories of software required to connect these many different systems.

Each of these categories offers many competing APIs and services. Figure 4 highlights some of these APIs and services.

In the network services category, a number of network services exist, including electronic messaging, database, directory services, and others. The complexity is intensified because many of these services have several available implementations, with different APIs. Consider electronic messaging, for example. At least four competing standards exist for messaging APIs: vendor independent messaging (VIM), Microsoft Messaging API (MAPI), CCITT Standard X.400 and Novell Message Handling Services (MHS).

Figure 3. Three categories of enterprise software infrastructure



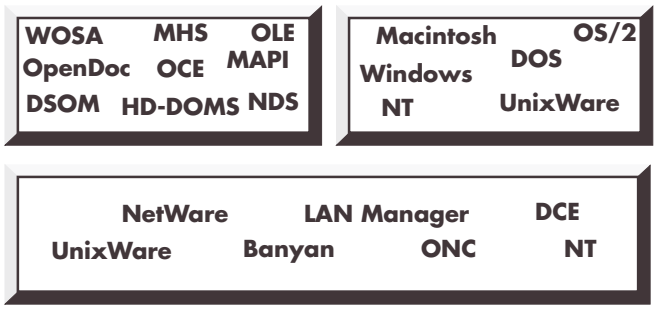


Figure 4. A mix of networks and client operating systems

In the client operating systems category, developers need to select the appropriate operating system for particular applications. Some of these operating systems include MS Windows, Macintosh, UNIX, OS/2 and DOS.

At the network category, multiple network standards can deliver network services to the layers above. Appendices A through D describe in detail the technologies that comprise this layer.

Each category offers a wide array of valuable choices. Each single API or standard can be a critical technology essential to the construction of a particular application. The challenge today is to leverage, unify and integrate these choices to produce real-world, mission-critical network applications. This is the infrastructure on which AppWare rests.

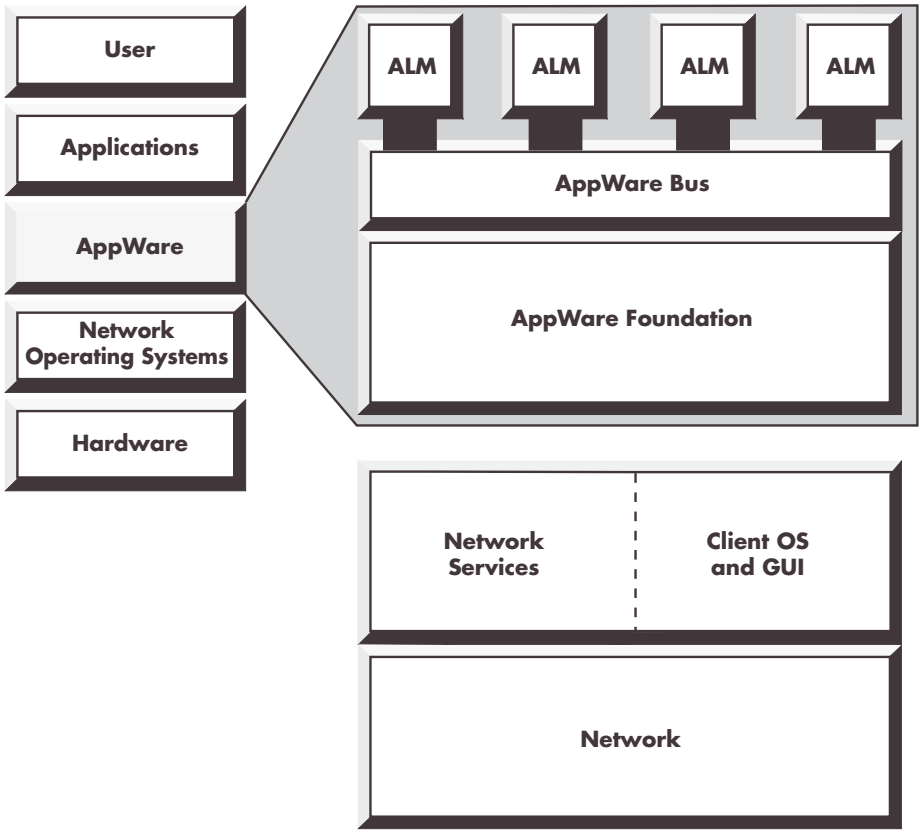
AppWare's Components

Just as a set of blueprints lays out the plans and components needed to construct a building, a computing architecture lays out the conceptual model required to organize a particular system. In addition, a computing architecture identifies the system's components and their relationships. Figure 5 illustrates the components of AppWare.

AppWare is based on the concept of leveraging existing technologies. Thus, the AppWare architecture builds on top of all major client operating systems and most major distributed services, including file systems, shared printing resources, document management, imaging, telephony, digital multimedia and directory management services.

In the past, many of the functions these services supplied were hard-coded into applications. This doubled the effort required to deliver an application. Early GUI programs had to be built directly to the graphics hardware, which proved to be an extremely time-consuming task. Developers soon learned to create libraries of the common, reusable functions, leading to the development of specialized GUIs. Though applications with GUI interfaces were easy to use, their development cost could still be prohibitive. It was not until the Apple Macintosh pioneered the availability of a widespread, consistent GUI, that applications built around this technology began to gain momentum.

Figure 5. The AppWare architecture



Database technology development has experienced a similar progression. At first, developers wrote applications directly to the file system, and the data was managed differently for each individual application. The format of the data was also application-specific, which made it difficult to share data among applications. Again, the next stage was to construct common libraries for data storage and access. This refinement allowed particular systems of programs to share data. However, as data storage and access technology continued to become more widespread and complex, corporate MIS were unable to keep up with the need to deliver data over the network to multiple users across multiple platforms. In response, database vendors began providing cross-platform database engines that a number of applications share over the network. Today, those same database vendors are trying to provide standards, such as Structured Query Language (SQL) and Integrated Database API (IDAPI), that address the issues inherent in managing data simultaneously across multiple databases and data models.

A broad variety of vendors providing a wide array of services, as well as multiple implementations of these services, has created the need for the first AppWare component, called the AppWare Foundation.

The AppWare Foundation

The AppWare Foundation hides the complexities of networks, local operating systems, and GUIs from developers who write line-by-line code using 3GLs, such as C, C++, COBOL and Pascal (see Figure 6). The AppWare Foundation provides a consistent, standard set of APIs that allows developers to access the services provided by both the network and the local operating system. Using the AppWare Foundation, developers can create cross-platform network applications without compromising system performance, application functionality and compatibility with existing and emerging technologies.

The AppWare Foundation includes the Universal Component System (UCS) technology which Novell acquired with the purchase of Software Transformation, Inc. Other components include the CPI-C interfaces for host connectivity and the X/Open distributed transaction processing APIs, which are supported by Tuxedo. The AppWare Foundation also supports multiple compound document architectures, such as Apple's Compound Document Architecture and Microsoft's Object Linking and Embedding (OLE) architecture. Appendix E describes these technologies.

Applications written to the AppWare Foundation can connect transparently to existing network services. An application programmer need no longer be aware of where a service is located or how it can be accessed. The programmer simply chooses the application's target platform, and the AppWare Foundation builds the necessary code to connect the application to the native services of that platform. When the developer recompiles the same code for a different platform, the AppWare Foundation implements the appropriate native services for the chosen platform, again without involving the programmer. For example, using the AppWare Foundation, a programmer writes the same code whether he or she is accessing files from an application on a Macintosh, MS Windows or a UNIX platform.

Application Portability

Using the AppWare Foundation's API set, the developer writes the application code only once and recompiles it to run on multiple platforms. This kind of portability for desktop applications is critical because many organizations have a mix of desktop platforms. Even organizations that have settled on a

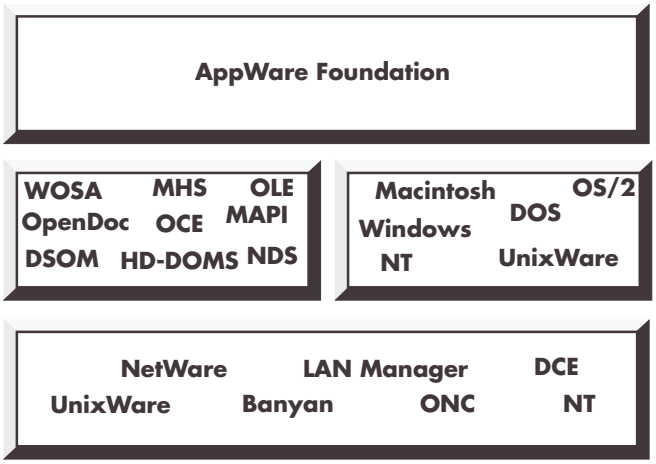


Figure 6. The AppWare Foundation is independent of network services, client operating systems, and GUIs

single desktop platform may need such portability to accommodate and take advantage of new technologies as they emerge. The AppWare Foundation provides portability across a wide variety of platforms, letting businesses take advantage of new technologies while preserving current system investments.

The goal of platform portability toolkits is to provide the same functionality across different platforms. As simple as this may sound, it is actually quite difficult. For example, almost all applications allow users to enter text in one form or another. However, text-editing ranges from basic editing, such as modifying the data fields in a forms package, to highly sophisticated editing, such as revising a document using a publishing system. When features are available on all platforms, as is the case with simple text entry fields or pull-down menus, compatibility problems seldom arise. The real question is how a multiplatform development environment should handle features not available on all platforms, such as multiple fonts in a text box or undo capability.

Platform portability toolkits typically take one of two approaches: they provide only common features (least-common-denominator toolkits) or they provide all the features (superset functionality). Taking the least-common-denominator approach is very efficient, but does not meet specific functionality requirements. However, providing a complete superset of the features available on each platform is probably impossible.

The AppWare Foundation does not provide portability by dropping to the lowest common denominator of all the supported environments. Rather, it identifies real-world application requirements for given areas of functionality. Upon establishing such requirements, functionality was added to those platforms that required it. While this is a superset approach, AppWare Foundation stops short of providing a 100 percent superset, concentrating instead on the functionality that commercial developers are most likely to need. Feedback from current users indicates that the AppWare Foundation is one of the most robust technologies available.

Most horizontal business applications can be coded exclusively to the AppWare Foundation, ensuring maximum portability and transparency of access to network services. However, the architecture is flexible and will allow programmers to drop to the operating systems' native APIs to gain more direct access to the system software or hardware as needed. Therefore, applications can still take full advantage of the unique functionalities and native services of local operating systems and networks.

In addition, programmers are free to use the platforms, compilers, linkers and debuggers of their choice. The AppWare Foundation is compatible with Symantec, Borland, Microsoft, MPW, Lightspeed, SABER and GNU compilers, and with Multiscope, Codeview, and SADE debuggers.

Underneath the AppWare Foundation exists the Common Request Broker Architecture (CORBA) specified by the Object Management Group (OMG). CORBA provides an infrastructure that allows objects to communicate, and is independent of specific platforms and languages used in the implementation of the objects. The CORBA architecture is described in Appendix F.

Benefits of the AppWare Foundation

The AppWare Foundation provides application developers with a complete set of APIs for implementing enterprise business applications using 3GLs. The benefits of the AppWare Foundation include the following:

- A single API set for different operating systems and networks
- Portability of the applications built to the AppWare Foundation
- Transparent access to network services
- High application performance
- Support for the evolution to distributed objects

For additional information about the AppWare Foundation, please refer to the *Novell AppWare Foundation* White Paper.

The AppWare Bus and AppWare Loadable Modules

Although the AppWare Foundation simplifies the application development process, it supports only those developers who write applications with 3GLs. In other words, one must still be a well-trained programmer in traditional programming languages to create applications based on the AppWare Foundation set of APIs. Corporate and vertical software developers using 4GL and 5GL tools require a much more rapid, efficient development platform.

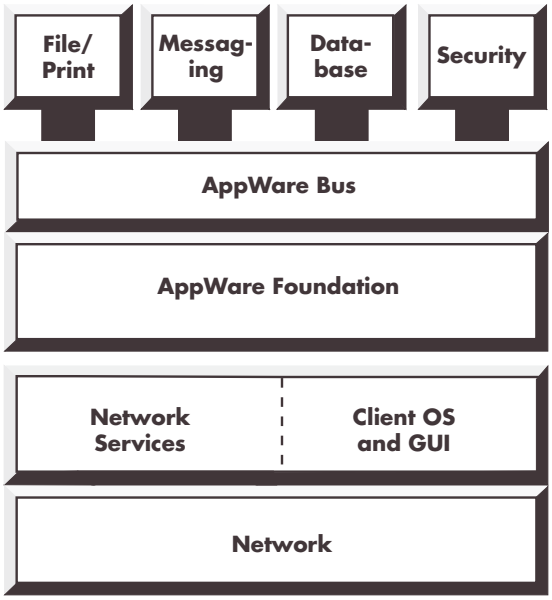
In these rapidly changing environments, access to prefabricated software is the key, thereby transforming the creation of applications from an art form to an assembly line production model. Building software with 3GLs is similar to building a car from raw metal, glass, rubber, and plastic. Building software using reusable modules is parallel to building a car from ready-made parts, such as an engine, wheels, seats, instruments and a steering wheel.

The AppWare Bus is a software engine that does for applications what the hardware bus did for personal computers — namely, it manages and coordinates the interaction of prefabricated, plug-in software components called AppWare Loadable Modules (ALMs) (see Figure 7).

ALMs are software objects that provide access to the functionality provided by both local operating systems and network services. ALMs can range from simple graphical utilities and spreadsheet modules to network services such as database and messaging. Business application developers can create new applications quickly and easily by linking different ALMs. The Novell Visual AppBuilder™ tool is designed specifically for this task. This tool is described in more detail in the following pages. Developers can also use other 4GL and 5GL tools that are compatible with the AppWare Bus to create new applications.

ALMs are large-grained, high-level software objects, which means they are much larger and more automatic than, for example, C++ classes. In a typical business application, a developer may use only 25 different ALMs to create all the needed functionality. Using C++ classes, as many as 500 to 1000 different classes can be used to create the same functionality. ALMs are more intelligent and functional than lower-level software components such as classes, but are smaller than today's massive horizontal applications (See Figure 8).

Figure 7. AppWare Loadable Modules



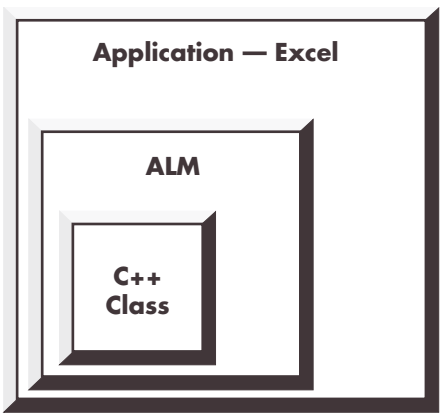
Since ALMs plug into the AppWare Bus, they can communicate and work together—even when created by different programmers.

Corporate developers typically have many in-house projects with overlapping components, such as custom reporting applications for different departments that must access the same data. Using ALMs, corporate developers can link the appropriate modules to create applications, reusing existing modules and leveraging each other's work.

Creating ALMs

Novell has already created a number of basic and network ALMs, such as MHS and Btrieve®, and is actively working with a number of ISVs to provide additional ALMs. For example, Cheyenne Software is creating ALMs for imaging and document management. Various third parties have already built ALMs for accessing Oracle and Sybase databases. In addition, MIS programmers may want to create new ALMs to provide key elements of functionality with

Figure 8. ALMs compared to other software components



information systems. Then, business application developers can link ALMs without having to understand how they were built.

ALMs are built using 3GLs and the AppWare Foundation. ALMs can be created with almost any Microsoft, Borland, Symantec, or MPW compiler. The *Novell ALM Construction Kit* provides the interfaces necessary to plug ALMs into the AppWare Bus.

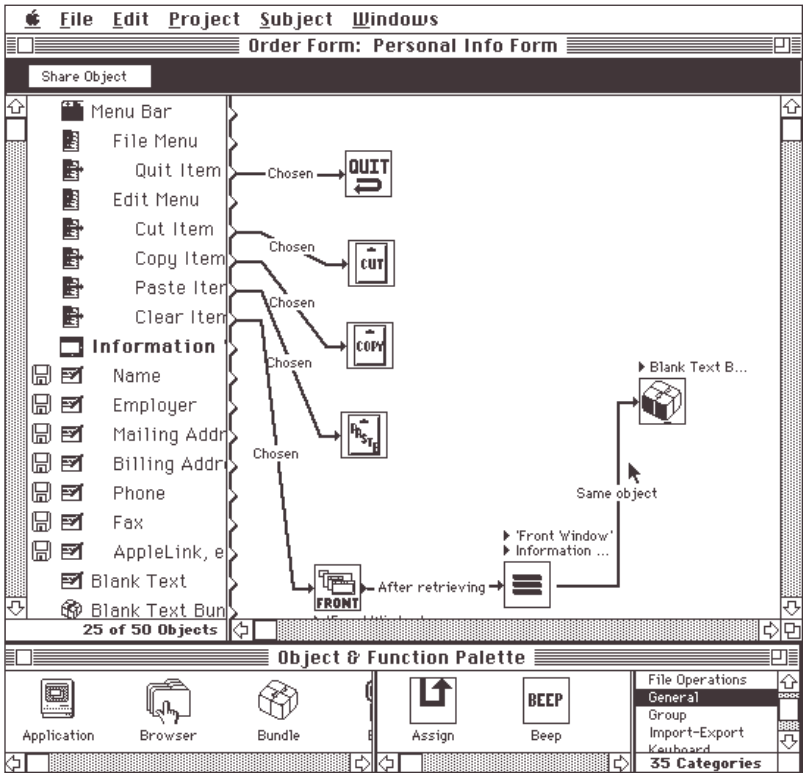
AppWare eliminates at least two risks for custom software providers: time-to-market and environment selection. The ALMs available on the AppWare Bus can significantly reduce application development cycles. Also, AppWare's availability for most major commercial desktops reduces the cost of a porting effort to a simple recompile.

Novell Visual AppBuilder

Novell now offers a high-level programming tool that has been tightly integrated into AppWare. The Visual AppBuilder tool provides an environment for rapid application development of network and standalone applications. Visual AppBuilder allows applications to be constructed by selecting icons that represent different ALMs. Then, developers use on-screen links to create application logic (see Figure 9). Visual AppBuilder provides a 5GL-level development environment, enabling programmers who do not necessarily know the details of 3GL development to rapidly create full-featured, reliable applications.

The range of applications developers can create is limited only by the range of ALMs provided by Novell and third parties. This makes Visual AppBuilder an ideal, easy-to-use tool for developing corporate and vertical business applications that keep pace with today's changing business environment.

Figure 9. Novell Visual AppBuilder session on a Macintosh



For example, consider a developer who wants to provide an easy-to-use automated telephone directory system for a global corporation. Suppose the directory information is stored in an Oracle database and the users of this program are non-technical, so the application needs a multimedia front end. In addition, the available hardware offers automatic answer and dialing through the telephony services on the network.

Using a 3GL tool, it would take several years and many developers to create such an application. Visual AppBuilder enables a few developers to build this application in only a week. A developer would simply connect graphically the Oracle database ALM, the multimedia ALM and the telephony ALM to create the new application without writing traditional line-by-line coding. Because ALMs are based on the AppWare Foundation, the application can be ported quickly to any common operating system and network.

ALMs make available all the services provided by the network and the local operating system. With ALMs for directory services, security, licensing, software distribution, telephony, work flow, imaging, multimedia and even office productivity, a new world of networked applications is now open to corporate and vertical software developers.

Benefits of the AppWare Bus and ALMs

The AppWare Bus, ALMs and Novell's Visual AppBuilder benefit corporate developers, smaller ISVs, system integrators and Value-Added Resellers (VARs) whose software needs exceed horizontal productivity applications. Some of these benefits include the following:

- Rapid "plug-and-play" model for application development
- Network services available to all developers
- Reliable, powerful applications for corporate MIS, vertical software developers and system integrators
- 4GL and 5GL development tools, rather than 3GL tools
- Use of 100 percent open and extendible through new ALMs
- Portable over major operating systems and networks

For additional information on the AppWare Bus, ALMs and Novell's Visual AppBuilder, please refer to the *Novell Visual AppBuilder* White Paper.

Integration with Third-Party Tools

AppWare's open architecture allows application developers to integrate the AppWare Foundation and AppWare Bus with a variety of third-party tools to provide additional functionality. For example, using AppWare Foundation APIs with Microsoft's Visual C++ can increase the efficiency of creating cross-platform applications. Novell's goal is to work closely with third party tool vendors to accomplish the following:

- Provide access to network and other services through the AppWare Foundation, so developers can deliver more sophisticated applications while writing less code.
- Extend the base of ALMs, allowing developers to quickly construct business solutions by combining ALM building blocks.
- Allow 4GL tool vendors, such as Gupta Technologies, Powersoft and Easel Corporation, to access the functionalities of ALMs.

Using AppWare: An Example

Suppose a developer wishes to create a networked document management application that will, among other things, route documents to fax servers and electronic mail servers. The application is designed such that the document destination handling is performed on a file server, but the user interface and editing are handled at the client workstation. Consider the development process using three different approaches of software development tools.

First Scenario

Using a relatively traditional set of application development tools, a developer must write core application modules that perform the document editing, as well as other software modules. Developers would have to do the following:

- Design protocols to establish communications between the client workstation and the machine providing the routing services
- Handle the construction, buffering, and sending and receiving packets at both ends of this process.
- Handle error detection, resending and acknowledging packets.
- Design some mechanism for locating the service provider, perhaps as rudimentary as having the user enter the name or address of the remote machine.
- Design an algorithm for establishing and verifying the identity of the client, possibly involving maintaining passwords or keys.
- Design and implement a minimal command language to enable the processes to perform such functions as opening a file remotely (for reading and writing).
- Design a user interface and implement the routing services on the target server machine and design the user interface code on the target client machine.

- Port the code for both the client and the server to several different environments, each one having specific communications code, graphics code and operating system calls.

This scenario provides innumerable challenges to the programmers and takes months or years to implement. It also presents a challenge for testing and maintenance, which grows exponentially with each new version of the product.

Second Scenario

In the second scenario, the developer of the same document management application will still have the same editing and routing code to implement, but can limit the communications and housekeeping modules. In this scenario, the developer may use, for example, Remote Procedure Calls (RPCs) to establish communication and perform the authentication, RPCbind or NIS to locate the service, and some GUI tools to build the user interface. The developer must write a specification file for RPCgen, determine whether a datagram or guaranteed message service is necessary for this specific application, and provide a means to specify the acceptable transport service, such as a configuration file. After writing the service code for the server and the user interface and writing the front-end code for the client workstation, the developer ports the code to different platforms, such as Macintosh, MS Windows and UnixWare.

Third Scenario

In the third scenario, the developer uses Visual AppBuilder or another AppWare-compatible tool to rapidly build his application, on both the server and the client sides. The developer designs and implements the solution by graphically linking ALMs. The ALMs, for example, might be a text editor, a document tagging service, fax services and electronic mail services. The ALMs are represented by icons and the application is created by linking the appropriate icons.

The ALMs the developer uses have already been written to the AppWare Foundation. These ALMs rely on the underlying location broker to locate the fax and e-mail services, to verify and select the appropriate available transports, and to authenticate the user. The developer uses the windowing system provided by GUI ALMs to build the user interface and recompile the application once for each target environment.

Conclusion

Building an infrastructure is an ambitious undertaking which can never be totally completed. The construction of an interstate highway system provides a good example. The expense and effort were considerable, but the benefits reaped from making new businesses and opportunities possible far surpassed the outlay in resources.

Similarly, a development infrastructure like AppWare can enable applications that might otherwise be impossible to bring to market. By providing an underlying collection of high-level, advanced services and the communications necessary to link service consumers and service producers, AppWare makes the construction of distributed applications far simpler than ever before.

Instead of the five- to eight-year effort it has taken to bring groupware products to market, AppWare can cut the time required to deliver such products to less than two years. The time savings results from the ability to use predefined services for database, calendaring, messaging and the like, without having to construct them from scratch. Also, the ability to use the network without having to subdue the protocol, communications and handshaking issues saves additional time.

By adopting AppWare as a development framework, application developers gain more than time. They gain:

- Access to a broad range of platforms without requiring multiple instances of their code for every platform they wish to support, eliminating the support and maintenance headaches.
- Access to a rich set of services along with transparent network access, thereby cutting application development time and extending the kinds of application problems they can solve.
- Control over a set of application tools and methods enabling developers to easily create client- and server-side modules.

All this adds up to increased developer productivity, better application flexibility, improved leverage of effort and broader markets for software.

By incorporating AppWare-based applications into their networks, users benefit from AppWare. Users gain:

- More broadly available network services. The power and flexibility of AppWare-based applications will make distributed services more broadly available.
- More flexibility and ability to change in the face of shifting needs and requirements. The customized nature of AppWare-based applications enables this flexibility.
- More extendible applications. The open-ended nature of ALMs makes such applications far more extendible than before and enables them to take advantage of new or enhanced services.

All this adds up to increased user productivity, more responsive and capable applications and rapid incorporation of technological changes and advances.

Why Novell?

The primary reason Novell is taking this bold step is to break the application backlog that threatens to slow the growth of the networking industry. Other important reasons include responding to customers' demands for more advanced, distributed applications; the desire to make networking completely central to information systems; and the fundamental need to improve the technology used to solve business problems. Novell believes its work and expertise in laying the groundwork for an application development infrastructure, and its openness to working with partners and customers, make Novell uniquely qualified to meet this challenge.

Although Novell did not focus on the development marketplace in the past, it is uniquely positioned to deliver a development infrastructure. For the last

ten years, Novell has been building some of the most advanced networking solutions available in the marketplace. Novell offers support for more networking topologies and technologies than any other vendor. Likewise, Novell supports more client and service platforms than all other vendors, ranging from desktops to mainframes, all of which can interoperate freely within the NetWare environment.

Novell also offers the broadest range of integrated internetworking solutions, for both wide-area and local-area access, across most available communications technologies. These offerings comprise the most complete communications and interoperability solution available from any vendor. In summary, Novell has the networking and interoperability coverage needed to support an infrastructure.

Just as Novell has shielded complexity and raised the level of network productivity in the past, Novell can shield complexity and raise the level of productivity for networked applications development and deployment.

No single company can provide a complete infrastructure, and Novell's unparalleled partnerships and programs play an important role in the development of a complete infrastructure. Novell has always been a partnering company; Chairman and CEO Ray Noorda is credited with coining the term "coopetition" to reflect Novell's willingness to cooperate with its fiercest competitors. Novell has also been an impetus for forming industry-wide groups, like the Technical Support Alliance™ (TSA), to bring vendors together under a single support umbrella, and avoid the finger-pointing exercises that network troubleshooting can so often cause.

Novell offers testing and compliance programs, such as "Yes It Runs With NetWare" for NetWare-compatible products. Novell also has one of the broadest education and certification programs for networking specialists in the world, the "Certified NetWare" professional programs. In addition, Novell offers options for training, service and support that continue to be widely emulated throughout the computing industry. Novell also maintains relationships and alliances with key consulting firms, major platform vendors and customer groups, to stay in close touch with industry trends, technologies and customer requirements. These partnerships and programs demonstrate Novell's ability to enlist broad support for an infrastructure, as it works closely with all the key players needed to stay ahead of emerging technologies.

In addition, Novell offers a wide range of distributed services to multiple client platforms through its NetWare and Unix products, from file and print, to database, messaging and directory. In the next 18 to 24 months, Novell will add support for services to provide electronic software distribution; software license metering and monitoring; imaging; telephony; document management; and multimedia from internal efforts and with joint development with companies like Imagery and Fluent Technologies (multimedia). These services highlight the ideal platform Novell offers to supply the advanced services that an infrastructure can deliver.

Novell has the ability to excel at enterprise networking and interoperability; it has the partnerships and programs needed to support enterprises; and it provides the richest set of distributed services. Therefore, Novell is pushing

forward to build an application development infrastructure to leverage these assets.

Novell clearly recognizes that the process of realizing the AppWare architecture is a lengthy, difficult and labor-intensive task. Novell cannot tackle this effort alone; significant input and cooperation from partners and developers has been and will be required. Even so, the potential benefits are enormous, and the rewards are significant.

To prove its depth of commitment to AppWare, Novell will publish the AppWare interfaces as they are defined. Novell also intends to keep the AppWare development tools open to all interested third parties, and will augment the technologies and interfaces that AppWare accommodates based on customer requests and market demands. In addition, Novell is committed to using AppWare for its own internal development efforts.

Novell is committed to providing complete education, service and support for all components of the AppWare initiative, and to providing copious background and training materials about the AppWare architecture. Since the value of an infrastructure is measured only by the way it is used, Novell's primary goal is to build an infrastructure that suits the needs of the developers who use it.

Appendix A: The NetWare Operating System

NetWare is Novell's network operating system. While many other networking environments are discussed as if they were network operating systems, they are in fact add-ons to existing general purpose operating systems such as UNIX or OS/2. Novell's NetWare was developed as a specialized system for providing network services. Its heritage as a service platform, enables NetWare to provide a highly optimized platform for hosting services, as opposed to general purpose applications such as spreadsheets.

NetWare also exports network services to applications running on other platforms. For example, the NetWare Directory Services allow users and applications to find resources on the network. The components of NetWare that enable distributed services are described in the following sections.

The NetWare Multiprotocol Architecture

NetWare's protocol engine allows NetWare to support multiple transport protocols. More importantly, this protocol engine allows applications to access distributed services through several protocols. This access is crucial to supporting a heterogeneous enterprise network.

STREAMS and Open Datalink Interface

STREAMS was created by Dennis Ritchie of AT&T Bell Laboratories, as an input-output system for implementing terminal drivers and network protocols. This system has become an integral part of UNIX System V. A stream is a collection of modules providing a head at one end and a driver at the other end. The modules between the head and tail can provide transport protocol stacks, filters and data routers.

The architecture of STREAMS allows a stream to be multiplexed at the head to any number of processes, and at the driver to any number of drivers. In the context of networking, STREAMS provides the mechanism for allowing an application or service to communicate over multiple transport protocols, by connecting through a STREAMS multiplexor to several stream heads. STREAMS also allows a protocol stack to multiplex to several hardware drivers, and therefore operate over multiple media, for example, Ethernet and Token-ring.

In the case of NetWare, the interface specified for drivers to connect with STREAMS-based protocol stacks is called the Open Datalink Interface™ (ODI). ODI has become the standard for PC-LAN network interface card drivers and Novell has recently augmented the specification to provide for 32-bit drivers that will work in either 16-bit or 32-bit environments. Thus, a driver developed for the NetWare server will work in the DOS, OS/2, MS Windows and UnixWare environments.

The Interfaces: IPX™/SPX Sockets, TLI and TIRPC

The Transport Level Interface (TLI) is an interface to a STREAMS-based transport provider. The interface is defined to be as independent of the transport provider as possible, yet allow access to particular functionalities of a

given transport. On NetWare, TLI can be used to interface to the AppleTalk Apple Data Stream Protocol (ADSP) stack, the Transmission Control Protocol/Internet Protocol (TCP/IP) stack, as well as Novell's own Sequenced Packet Exchange/Internetwork Packet Exchange (SPX/IPX). Access to specific characteristics of a transport are handled in a standardized manner, enabling the application or service to manage connections over a wide variety of transports. Novell provides the TLI on the NetWare server, DOS, MS Windows, OS/2, and UnixWare.

Remote Procedure Calls (RPCs) provide a standardized inter-application protocol. The model of one procedure within an application calling another is thereby extended to include invocation of procedures running on other systems on the network. RPC masks the intricacies of the communications code from the developer of a distributed application. Novell provides an RPC mechanism based on and interoperable with UNIX System Laboratories' Transport Independent RPC.

The RPC technology must also address problems regarding data types on differing hardware platforms. An example is the difference in integer representation on the Motorola 68000 microprocessors and the Intel i8X microprocessors. The Motorola processor stores integers with the most significant byte first; the Intel processors store integers with the least significant byte as the first. The RPC mechanism deals with this challenge through the eXternal Data Representation (XDR) format. XDR specifies a standard by which data can be represented in a machine-independent format. Note that XDR can be used by itself to provide machine-independent data representations in contexts other than RPCs.

Transport Independent RPC (TIRPC) allows a service or application to be built such that it will accept connections over more than one transport protocol. TIRPC leverages the Transport Level Interface to achieve transport independence. Thus, a service can provide its functionality to other services and applications on the network, regardless of the transport protocol connecting them.

NetWare Authentication

Authentication allows objects on the network to "prove" their identities to one another. As such, authentication is the basis for all network security. Once identity is established in a network session, the system can determine whether the user has access to a network resource by checking an access rights list associated with a given object.

The authentication services provided by networking software offer applications an infrastructure so they can set up authenticated sessions and verify that the user connecting to their service has rights to access the information it accesses.

Authentication is a network service; other applications can use NetWare's Authentication Services infrastructure and APIs to set up authenticated sessions between their own services and clients. In NetWare, authentication is session-oriented. A client's signature, the basis of authentication, is valid only for the duration of the client's login session. The signature itself is never transmitted across the network.

NetWare's authentication mechanism is based on RSA technology, a public-key encryption model named after its inventors, Rivest, Shamir and Adleman. The RSA method eliminates the need for transmittal of unencrypted keys.

Once an authenticated connection to the network has been established, all other authentication takes place in the background without need to obtain additional user passwords.

The interfaces for authentication are straightforward. There are calls to login to the network and to authenticate to servers on the network based on connection IDs.

NetWare Basic Network Services

After an application or user authenticates itself to the network, it can take advantage of the other basic network services, including some Directory Services, Time Services and the file system services.

NetWare Directory Services

As more data is distributed across networks, users find it increasingly difficult to determine what is available on the network and how to get access to it. Directory Services provides applications with a means of finding things on the network. A directory is an object database that provides information on the resources available on the network and how they can be accessed.

NetWare Directory Services (NDS) is a global, distributed and replicated database. It is based on parts of the CCITT X.500 standard. The Directory Access Services perform the work of accessing the directory and returning information. The basis for the NDS Access Services operations is the set of abstract services described in the CCITT X.500 Directory Recommendations. The NetWare interfaces are derived from these abstract service definitions.

NetWare Time Services

The purpose of a time service is to synchronize the clocks of computers on the network with the Universal Time Coordinate. Synchronization is critical on a network because the consistency and integrity of the distributed directory database depends on a coordinated time.

The system clocks on the network are divided into clients and servers. Servers synchronize with each other and clients match time with the servers. The interfaces for accessing this coordinated time include ANSI standard interfaces.

NetWare Universal File System

The NetWare File System provides a universal file server engine upon which a number of file system protocols have been implemented. The directory structure of the NetWare server provides for multiple Name Spaces, allowing the correct information to be kept for a file so that it can be presented as an Apple Macintosh, ISO FTAM, UNIX NFS, OS/2 HPFS, or DOS file. The DOS file system access is built into the NetWare operating system. Other filing protocols are provided by adding the appropriate NetWare Loadable Modules™ (NLMs).

It should be noted that access under each of these models is "just as if" the access were under the native server environment, so that a UNIX client to the NetWare file services would operate on the file with the standard UNIX interfaces.

Appendix B: Open Network Computing (ONC)

The Open Network Computing model was largely defined by Sun Microsystems and includes what has become the standard distributed file system protocol in the UNIX environment—NFS. Much of the ONC technology has since been included in the UNIX SVR4.X offering from UNIX System Laboratories, and is available from Novell as part of the UnixWare Application Server. The following are the components of ONC.

ONC Transport Protocol Architecture

The ONC networking model, like NetWare, is built on a STREAMS-based protocol environment, with TLI and TIRPC as key developer interfaces. The overlap in supported protocols is also significant: both environments support TCP/IP, the ISO transport protocols, and in the UnixWare implementation of SVR4.2, IPX/SPX.

Streams

USL's UNIX SVR4 release merged much of the technologies represented in the UNIX System V Family with the other prevalent UNIX variant of the time, Berkeley System Distribution (BSD) 4.3. Also, Sun Microsystems had built the Sun OS on the Berkeley distribution, having added the RPC component and NFS to provide a basis for distributed computing. Sun redesigned its RPC to become transport-independent by implementing it using Transport Layer Interface (TLI), a STREAMS-based transport provider interface.

The Interfaces: TLI and TIRPC

The implementations of TLI and TIRPC in the ONC environment interoperate with Novell's network environment. Communications code written using these interfaces should readily port between the environments as well.

Kerberos Authentication

Kerberos Authentication is an add-on to this environment. It is not offered uniformly, but the underlying RPC mechanism has provision for adding extensions to use whatever authentication mechanisms are available. Typically, the authentication mechanism used is that available in the UNIX system implementation.

ONC Basic Network Services

The Network Information Service (NIS+)

NIS+ provides a naming service similar to that of NetWare's. It is based loosely on the X.500 standard. NIS+ was architected to interface with other naming services to provide gateway capabilities extending to other name spaces.

Network Time Protocol (NTP)

ONC supports the NTP, an Internet standard defined by the Internet Activities Board. This protocol provides the same functionality as the NetWare Time Services.

The Network File System (NFS)

The Network File System has become the de facto standard for distributed file systems in the UNIX environment. It is built upon the RPC protocols and uses the XDR standard data format to ensure that data can be stored and retrieved regardless of the machine architecture of the server or the client. NFS supports a subset of local file semantics for the access of remote files, allowing clients to access file systems or subtrees of file directories on remote systems. NFS is supported by Novell's UnixWare Application Server Product and is available as an add-on file access protocol to the NetWare server.

Appendix C: Distributed Computing Environment (DCE)

The OSF is a consortium of hardware and software vendors whose goal is to provide a common operating system and distributed computing environment for heterogeneous platforms. The DCE is comprised of a set of services that can be used individually or together to form the infrastructure for distributed computing. DCE categorizes the components into two classes: Fundamental Services, those services upon which developers will implement other distributed services; and Data Sharing services, which provide end-users with file sharing and printing capabilities.

DCE technology comes from the following vendors: Digital Equipment Corporation, Hewlett-Packard Company, The Massachusetts Institute of Technology, Siemens AG, Microsoft Corporation, Sun Microsystems, and Transarc Corporation.

The following sections describe the components of DCE.

The DCE Protocol Architecture

DCE does not specify how protocols are to be implemented. However, OSF has endorsed the use of the XTI interface, which is an X/Open® version of Transport Layer Interface (TLI). The DCE RPC mechanism has been implemented over XTI and gains transport independence as a result.

The Interfaces: Sockets and DCE RPC

In most DCE implementations, a form of the Berkeley System Distribution Socket interface is supported. This interface is not specified as a part of DCE, but is a lower-level element that is usually present.

DCE places great emphasis on its RPC mechanism as the primary vehicle for implementing distributed applications and services. The DCE RPC comes from Digital Equipment Corporation and Hewlett-Packard. A primary difference from the ONC TIRPC is in the data format. DCE uses the Network Data Representation (NDR) which operates on the receiver-makes-right principle. The differences between XDR and NDR preclude interoperability between the two RPC mechanisms.

Within DCE, RPC is viewed as a fundamental architectural element upon which all other elements of the architecture derive their means of distribution.

Kerberos Authentication

An integral part of DCE's security services, Kerberos is based on private-key encryption technology, and thus relies on the physical security of the authentication server. Private-key authentication mechanisms do not lend themselves to global or intercell authentication because there is no mechanism for protecting the key while it is being transferred. The key must remain "private" to maintain security.

Because of this, OSF plans to ensure that applications built to their environment are portable from Kerberos to public key authentication mechanisms such as the RSA mechanism used in NetWare.

DCE Distributed Naming Service

The DCE supports two name service models: a cell-based service, which has stringent performance requirements, and a second model that provides support for the global X.500 directory protocols. The OSF believes this dual model approach best serves the requirements of performance, as well as global interoperability across an enterprise. Developers use a common programmatic interface to access either cell directory services or the remote name servers. Note that the technology for these naming services came from two different companies; Digital Equipment supplied its Distributed Naming Service (DECdns) for the cell directory, and Siemens submitted its DIR-X X.500 service. Furthermore, OSF's DCE provides a common API across these services.

Time Service

The time service technology was supplied by Digital Equipment. The purpose of the time service is the same as in the NetWare environment; other services such as the distributed file system and authentication mechanism require that systems on the network have synchronized clocks.

The service is based on an algorithm that synchronizes the time between time servers and allows clients to get the time from the synchronized servers.

DFS—The Distributed File System

The Distributed File System technology for DCE, the Andrew File System (AFS), came from Transarc. The architecture is built around a cache manager on DFS client systems connected via RPC to the NFS protocol exporter. The NFS protocol exporter allows existing NFS and (PC) NFS clients to access the DCE file system. The file system is integrated with the environment services for authentication, naming, and time service.

Appendix D: Microsoft NT Advanced Services Networking

Microsoft NT's networking derives from its earlier efforts, MS-Net and LAN Manager. While LAN Manager was often referred to as a network operating system, it was really a collection of add-on applications and drivers to the OS/2 operating system environment. With the advent of NT, Microsoft has added networking to the operating system kernel.

The architecture of Microsoft's network support has evolved into something resembling the other major networking environments, DCE, ONC and NetWare, albeit in a more proprietary fashion. The following sections highlight the components of Microsoft NT.

The Microsoft NT Protocol Architecture

The protocol architecture of NT combines the earlier NetBEUI with STREAMS-based protocol stacks such as TCP/IP and IPX/SPX. All of the protocols have an interface to the transport driver interface (TDI). Interfaces with which application developers normally interface, such as NetBIOS and Sockets, then interface with the TDI to gain access to the protocol stacks themselves.

At the lower end of the protocol stacks, the network driver interface specification (NDIS) defines the interface the network interface cards.

Streams

As previously mentioned, NT provides a STREAMS environment which maps at its upper boundary to TDI and to NDIS at its lower boundary. Protocol stacks can be implemented as STREAMS-based drivers or as monolithic drivers to TDI, as is the case with Microsoft's implementation of NetBEUI. Unlike the other networking environments, TLI or a derivative is not supported. However, TDI has some of the characteristics of the Transport Provider Interface (TPI) as defined in UNIX STREAMS, which is the protocol of the STREAM head to which TLI is connected.

The Interfaces: NDIS, TDI, MSRPC and Win Sock

Microsoft's networking interfaces provide functional equivalence to those supported in the other paradigms. NDIS is analogous to network interface card driver specifications ODI and DLPI. The TDI interface provides some of the functionality of the STREAMS Transport Provider Interface (TPI).

MS RPC, a Microsoft derivative of DCE's Remote Procedure Call technology, may provide a pathway to DCE-based services, but it is unclear at this time whether NT has the required client support to authenticate itself to DCE environments.

Advanced Network Services Authentication

The authentication mechanism used in the Advanced Services networking for NT is the same used to authenticate users as they log on to a stand alone NT platform. The user or application may log on either to the workstation, or to a "domain." When logging onto a domain, the user or application gains access to the resource directories of all NT platforms within that domain.

Basic Network Services

Microsoft NT provides basically the same functionality as LAN Manager 2.0 did, albeit implemented more directly in the kernel of the operating system. Microsoft NT has a redirector and responder architecture which tie to its TDI interface to provide protocol independence.

The naming service provided is domain-based and is not a true global directory. Microsoft has stated that it will provide a path whereby it interoperate with OSF's DCE to provide robust enterprise-level networking services. It is difficult to determine the exact nature of how NT's networking model will evolve.

Appendix E: Compound Document Architectures

Compound documents are documents that contain different types of media like text, graphics, video, and charts. Constructing these documents requires that a user move between several different applications, each with a different user interface. This process can be tedious and is certainly not productive.

Complex documents which contain several media are becoming the norm rather than an exception. At the same time, the single user desktop model is evolving to a collaborative computing model as workgroups combine efforts to create complex documents.

A new usage model is required that simplifies compound document creation in a collaborative environment. Compound document architectures are being developed by a number of different vendors to provide a model for the interaction of users with documents composed of a wide array of content types. In these architectures, developers are able to break monolithic applications into independent components, sometimes called "applets" which can be assembled into a whole within the framework of the document architecture.

Apple's Compound Document Architecture

Apple's Compound Document Architecture is a model for the creation and storage of compound documents. Its approach is to decompose software into independent modules, or "parts," which can be flexibly combined in a variety of ways.

The Apple Compound Document Architecture includes support for editing "in place" without having to exit the primary document and cut and paste. The architecture provides support for drafts which aid in managing versions of documents. Apple's architecture offers interoperability between heterogeneous platforms.

Novell is working with Apple and other third parties to deliver Apple's Compound Document Architecture on the MS Windows desktop, as well as enable the architecture for client/server network environments. One facet of Novell's implementation will be the ability to share compound documents among heterogeneous desktops.

Microsoft's OLE 2.0

OLE stands for Object Linking and Embedding. OLE is Microsoft's infrastructure for embedding objects within documents. These objects are comprised of data such as text or graphics coupled with the application functionality to manipulate that data.

OLE 2.0 supports in-place activation, allowing objects within documents to be activated with in the containing window of the application. Objects under OLE 2.0 may contain other objects, and the contained objects may be operated on.

OLE provides Microsoft with the basic capabilities required to support compound documents.

Appendix F: Object-Oriented Programming

The goal of OOP is to provide a programming model which supports the reuse of standard software building blocks. The primary benefits are improved reliability of applications and improved productivity of developers. OOP also provides a more natural way to design applications because applications become models of the "real" world.

OOP is based on a few well-defined concepts:

Objects—Programs are built by putting together software components called objects. An object may model a "real" world component like a checking account or automobile. Objects have one or more attributes or fields that define the state of the object. Objects have behavior which is defined by a set of methods which can modify the attributes of the object. Objects have a unique identifier which distinguishes it from other objects in the software system.

Encapsulation—the details of the data structures and methods are hidden. The only access to the data within an object is through its associated methods. Isolating the details of the object keeps the impact of changes to the object from impacting other components of the program. The object is a black box.

Messaging—The way objects interact is based on a client–server model, where the client object passes the server object a request message with any information required for the server to process the request. The server object then sends the client a reply. The services an object provides are defined by the methods the object makes available for other objects to use. It is interesting to note that this request reply model lends itself to distribution of object over a network. The message from one object to another simply goes over the network instead of remaining on the local platform.

Data Abstraction—Object classes are defined as being the attributes and methods for a type. An example might be a graphical object class such as a circle, which had methods to print, display, and size itself. A developer defines the class circle once. In defining the class he must write the code for the methods to draw a circle on the display, to print a circle on the printer, and to change the size of the circle. The developer can then create an instance of a circle and give it a name in his program. The code for the methods is stored with the class and occurs once, but the attributes or instance variables are stored for each instance.

Inheritance—Data abstraction enables the notion of subclasses and superclasses of a class. A developer can create a new class by "inheriting some or all of the characteristics of a class and adding new attributes and methods. In the case of the circle mentioned above, a color filled circle class could be created by inheriting from the base class circle and then adding a method to fill the object with a color, and an attribute to store the color. In some languages it is possible to inherit from more than one parent class. This is called multiple inheritance.

Distributed Object Computing

Distributed Object Computing is a new computing model that merges distributed computing with object-oriented technology. Objects in this model are software components which provide their services over the network. As discussed above, in the object model of programming an object interacts with other objects by sending requests to them. The objects receiving a request process it and sends back a response message. The Distributed Object Computing model extends this notion by interjecting an object management system which brokers service requests and responses.

Object Request Brokers

A system component called the Object Request Broker (ORB) acts to locate service providers and convey requests to them from clients needing the service, thereby providing a mechanism for transparently accessing distributed services from within an application.It is the job of the ORB to locate the service and communicate the request to the service provider. The ORB also coordinates sending the reply back to the requester of the service.

The ORB establishes a contract with the client application and with objects providing services on the network. Essentially the ORB extends to the client the contract to locate objects providing the requested service and to see that the request is fulfilled. The ORB extends to the object supplying the service a contract to connect it with clients requesting its services. To clients the ORB extends a contract that it will get the request to a server who can fulfill it and that it will get the reply back to the client.

The Common Object Request Broker Architecture (CORBA)

The Object Management Group (OMG), a non-profit international trade association, has specified the Common Object Request Broker Architecture (CORBA). This architecture specifies an infrastructure that allows objects to communicate, and is independent of specific platforms and languages used in the implementation of the objects.

The goal of the OMG is to foster a framework within which software components can be used in a plug-and-play fashion. Under this framework, a company could purchase a software component and write applications using its services without becoming locked into it. When a better implementation becomes available, it would be possible to substitute the new one without modifying the applications calling upon the services.

The CORBA model provides for an evolutionary path to distributed object computing. CORBA objects need not be implemented in an object-oriented language. In fact, many of the benefits of object orientation can be delivered to existing systems because it is possible to encapsulate the existing system with an object interface. This is done by writing a class interface front-end.

It is not necessary to have a complete implementation of the object request broker on each system where there are objects residing. A Basic Object Adapter (BOA) is the component in CORBA which handles object reference, object invocation, and state related services to a set of object implementations. An ORB may have several object adapters each specialized for a given environment. In the case of legacy systems, the object adapter may provide

specialized communication services from the host environment to the ORB running on a NetWare or UnixWare server. The legacy functionality is wrapped with an object-oriented front-end. This allows all clients and service platforms to access the legacy system's functionality as they would any other service on the network. In fact, the clients of the service need not be aware that the object's implementation is on a mainframe at all.

Distributed Object Management provides the AppWare Enterprise Architecture with an evolutionary path from procedural client–server computing into the future of distributed object computing. Distributed objects provide increased support for reusability and as a result significant gains in the productivity of development staff.

AppWare Glossary

Application Programming Interface (API)—Means by which an application gains access to system resources, usually for the purpose of communication, data retrieval or other system services.

AppWare—Name given to a new layer of software from Novell. AppWare leverages today's popular operating systems, development tools, and applications, hides the complexity of the network, and delivers its value.

AppWare Bus—Software engine that manages and coordinates the interaction of software components called ALMs.

AppWare Foundation—A set of standardized Application Programming Interfaces (APIs) for both the local operating system and network services. The AppWare Foundation API provides 3GL (like C, C++, COBOL) application programmers with a common cross platform interface to multiple Graphical User Interfaces (GUIs), operating systems, and network services.

AppWare Loadable Module (ALM)—Software object that provides access to the functionality provided by both local operating systems and network services. ALMs can be written with 3GL or 4GL tools.

ALM Construction Kit—Software Developer's Kit (SDK) from Novell that provides the interfaces for developing AppWare Loadable Modules (ALMs).

Attributes—Technique for describing access to properties of files and directories within a filing system. For NetWare files, attributes include Read, Write, Create, Delete and Execute Only. For NetWare directories, attributes include Read, Write, Create, Execute and Hidden.

Attributed File System—File system that stores attributes as well as data. Usually, attributed file systems store multimedia data found, for example, in compound documents.

Authentication—Allows objects on the network to identify themselves to one another. It is the basis for all network security. Once identity is established in a network session, the system can determine whether the user has access to a network resource by checking an access rights list associated with a given object.

CCITT—Committee that recommends standards for communications equipment interfaces, communications protocols, and modem modulation methods.

CICS—IBM's Command Information and Control System (CICS) transaction environment. Supports Online Transaction Processing (OLTP) systems under MVS, OS/2, and AIX.

Client operating systems—The system software at the desktop. This is often extended to include the GUI system support, like the Macintosh Toolbox, Motif, or Microsoft Windows.

Client–Server—Application model where a client sends requests to a service and receives replies. Client and server are roles. At different times an application can take on either role.

Codeview—Microsoft's source-level debugger.

Collaborative Applications—Application which allow users to share their work product, thereby enhancing productivity.

CORBA—Common Object Request Broker Architecture defined by the Object Management Group (OMG). CORBA specifies an infrastructure that allows objects to communicate, and is independent of specific platforms and languages used in the implementation of the objects.

COSE—Common Operating System Environment. Initiated by DEC, HP, IBM, SCO, SUN, Univel and USL to create a common set of APIs across all versions of the UNIX operating system.

Datagram—A network transport protocol which delivers packets on a best effort basis.

DCE—Distributed Computing Environment, the networking model defined by the Open Software Foundation (OSF).

Distributed Application—Application which executes on more than a single system.

Distributed Services—Software components residing on the network that provide functionality to client applications.

DOMS—Distributed Object Management System. A piece of software which manages storage of object interfaces and communication between objects and clients accessing the object's services.

DSOM—IBM's Distributed System Object Model. Provides a CORBA object management system for distributing objects under their standalone System Object Model.

GNU—Cross-platform suite of publicly available C and C++ development tools for the UNIX environment.

GUI—Graphical User Interface.

Heterogeneous Environment—Computing environment where different hardware, operating systems, and GUIs must coexist.

Lightspeed—A C-language compiler for the Apple Macintosh.

Message Handling Services (MHS)—Novell's messaging protocol.

Messaging Application Programming Interface (MAPI)—Microsoft's messaging interface.

Multiscope—Symantec, Inc.'s source code debugger.

MOTIF—GUI software developed by the Open Software Foundation. MOTIF runs over the XWindows protocol, and is the standard for GUIs in the UNIX environment.

Network Application—Application that uses network services or distributes its execution across multiple networked systems.

Network Service—Software component that provides functionality which can be accessed over the network by other applications or services.

Network Service Independent—Interface that provides access to more than one implementation of a service.

Novell Visual AppBuilder—High-level programming tool which is tightly integrated with the AppWare architecture. AppBuilder allows developers to construct applications by visually linking icons representing ALMs.

OLE—Object Linking and Embedding, Microsoft's infrastructure for embedding objects within documents. Objects are comprised of data—such as text or graphics—coupled with the application functionality to manipulate that data.

Open Doc—Apple's Compound Document Architecture, a model for creating and storing compound documents. Its approach is to decompose software into independent modules, or "parts", which can be flexibly combined in a variety of ways.

ONC—Open Network Computing, Sun Microsystem's networking model. It includes the Network File System (NFS), the standard distributed file system in the UNIX environment.

ORB—Object Request Broker, a system software component that acts as an intermediary between clients and objects that provide services.

OSF—Open Software Foundation, a consortium of hardware and software vendors with the goal of providing a common operating system and distributed computing environment for heterogeneous platforms.

Service provider—A software component that provides functionality used by other applications or software components.

Software bus—System software that enables software components to operate in a plug-and-play fashion, much like a hardware bus that enables option cards to extend the basic hardware system.

Software License Server—A software component for metering usage of software on a network.

SOM—System Object Model, IBM's architecture for defining and managing binary class libraries.

Sequenced Packet Exchange (SPX)—Protocol through which two workstations or applications communicate across the network. SPX uses NetWare IPX to deliver the messages, but SPX guarantees delivery of the message and maintains the order of messages on the packet stream.

Standalone application—Application that executes completely on a single machine. Network services are not leveraged.

System Application Architecture (SAA)—A set of IBM-defined standards that provide a consistent environment for programmers and users across a broad range of IBM equipment, including microcomputers, minicomputers and mainframes.

Telephony—A network service that integrates computer and telephone networking technology.

Third Generation Languages (3GLs)—Procedural programming languages, such as C, C++, FORTRAN, and COBOL.

TIRPC—Transport Independent Remote Procedure Call, a technology that provides a standardized inter-application protocol. With TIRPC, one procedure within an application can call another, and can invoke procedures running on another system on the network.

TLI—Transport Layer Interface, an interface to a STREAMS-based transport provider. TLI is independent of the transport provider, yet allows access to particular functionalities of a given transport.

Project—Basic unit of organization for an application built with Novell's Visual AppBuilder.

Subject—Reusable "sub-project" or unit of an application built with Novell's Visual AppBuilder.

Object—Basic unit of encapsulation as defined within Novell's Visual AppBuilder.

Function—The behavior of a Visual AppBuilder object.

Transaction—Unit of work that is guaranteed to complete in its entirety or be rolled back.

Transaction Monitor—A system software component that supports distributed transactions by providing capabilities such as data integrity through a two-phase commit, as well as security through authentication and authorization features.

TUXEDO—UNIX System Laboratories' transaction monitor.

Ubiquitous Computing—Term that applies to everyday devices which have embedded computational capability.

UNIX SVR4.X—UNIX System V Release 4.X, the current version of the UNIX operating system offered by UNIX Systems Laboratories.

Workflow—Integrated technology in the network environment that enables workgroups to share documents and work collaboratively.

WOSA—Microsoft's Windows Open Services Architecture, a model for connecting MS Windows applications to back-end services.