Novell AppWare Foundation

White Paper

July 1993

In today's business environment, software development organizations continue to strive toward one major objective: increased developer productivity. Every organization finds it nearly impossible to stay on their development schedule, keep up with the needs of their users and keep the development budget under control.

The heterogenous nature of today’s computing environment requires developers to maintain versions of any given application for multiple platforms. Having an application that ran on several platforms was considered a competitive advantage in the ’80s. Today’s software marketplace *demands* that applications run on several platforms.

This portability issue is further complicated by the recent proliferation of platforms, technologies and development environments. Organizations must choose from a variety of operating systems like DOS, UNIX, OS/2 and NT; different graphical user interfaces (GUIs) like Macintosh, Microsoft Windows, Motif and OpenLook; and numerous networking conventions including NetWare, AppleTalk, and TCP/IP. The issues become more complex every day.

The continuous need to increase productivity and the ever-increasing complexity of cross-platform portability have motivated many software development organizations to look for a single development environment that addresses both challenges. Novell recognizes the challenges developers face, and has designed AppWare, a system for developing network applications. Figure 1 shows the AppWare architecture (see page 3 of the hard copy).

What Is the AppWare Foundation?

The AppWare Foundation is one of the two major components of the AppWare environment. The AppWare Foundation provides 3GL application programmers with a common, cross-platform set of application programming interfaces (APIs) to multiple GUIs, operating systems, and network services. The AppWare Foundation allows developers to maintain a single-source base for all development platforms.

The AppWare Foundation is comprised of the Universal Component System technology which Novell acquired with Software Transformation, Inc. Other components include the CPI-C interfaces for host connectivity; the X/Open® distributed transaction processing APIs, which are supported by Tuxedo; and support for Apple's Compound Document Architecture and Microsoft's Object Linking and Embedding technology.

Novell's goal in developing the AppWare Foundation is to create a completely functional network development environment with an architecture that enhances programmer productivity and application performance. As the AppWare Foundation continues to evolve, it will provide even greater support for network services such as directory services, document management, messaging and database.

The AppWare Foundation Architecture

The AppWare Foundation architecture maximizes several characteristics developers require to successfully create today's network applications. These include:

- **Application performance.** Applications based on the AppWare Foundation provide the same level of performance as applications based on native implementations.

- **Toolkit functionality.** The AppWare Foundation provides developers all the functionality they need—GUI support as well as support for operating system services and application connectivity.

- **Toolkit modularity.** The AppWare Foundation architecture can be scaled, based upon the specifications of each application.

- **Toolkit extensibility.** The AppWare Foundation toolkit enables developers to extend functionality by implementing features not directly provided by the toolkit.

- **Development migration path.** The AppWare Foundation

architecture provides a method of mixing new and legacy code, since most development organizations cannot afford to migrate an entire application to a new development platform all at once.

- **Integration with other tools.** The architecture allows developers to work with their favorite third-party tools and easily integrate their own tools.

To incorporate these characteristics, the AppWare Foundation was implemented with a focus on six architectural elements (see Figure 2 on page 5 of the hard copy):

- Complete solution
- Superset of platform functionality
- Scalable architecture
- Native implementations for each platform
- Open system architecture
- Layered architecture

The following sections describe how the AppWare Foundation achieves these goals.

A Complete Solution

Currently, the portability challenge entails much more than simply supporting a variety of GUI platforms. In fact, meeting the need for cross-platform applications that maximize each platform's features is far more difficult than most GUI portability issues. And although connectivity issues have only recently become important, almost every new application coming to market needs a distributed architecture, focusing the attention of today's developers on connectivity issues.

As a result, it is important that the AppWare Foundation cover all the areas a programmer has to address with state-of-the-art, distributed applications. To meet these demands, the AppWare Foundation currently provides functionality in three areas: operating system or foundation series, connectivity and user interface series (see Figure 3 on page 6 of the hard copy).

Component Series and Components

AppWare Foundation is a collection of software modules called software components. These components share many characteristics with traditional code libraries, but there are important differences. Similar to a library, an individual component is a collection of utility routines that is reusable within and among applications. Components are implemented as modules using C and as classes using C++. Components encapsulate their routines with their associated data elements; some components rely on and inherit capabilities from other components.

The AppWare Foundation collects the various types of services available on supported platforms and divides them into three major areas of functionality: the Foundation Series, the Connectivity Series and the User Interface Series. Within these series, the AppWare Foundation currently provides more than 35 component families, such as Memory, File, Button, Window, Item, and Graph (see Figure 3 on page 6 of the hard copy). Each component family consists of core functionality and extensions to the core.

Core components contain the most efficient implementation of a component family's basic features. For example, the core component of the Edit Text Family supports features such as multiple lines of text; scrollable viewing; clipboard cut, copy and paste; single font in a view; maximum 32,767 characters; and left, center, and right justification.

Extensions are implementations of a family beyond these core functions. The MultiFont Extension of Edit Text, for example, supports multiple fonts in a view. Some component families, such as Graph, provide a great number of extensions. Other families, such as Font, currently have no extensions. Extensions allow developers to scale the size of AppWare Foundation to provide only the necessary functionality, therefore increasing performance.

The Foundation Series

The Foundation Series provides developers operating system services such as memory management, data management, file management, font selection, application internationalization, and device management.

In the area of memory management, for example, AppWare Foundation provides routines to bypass common memory module limitations, such as 64KB segments and limits on available handles. The Memory Component provides routines for allocating, locking, unlocking, resizing and deallocating memory from the heap (see Figure 4 on page 8 of the hard copy).

In file management, AppWare Foundation provides standard I/O functions, file and directory management, resource management facilities, and a way to deal with user preferences in a platform-independent manner.

To support flow of control issues, AppWare Foundation provides an error-handling facility and incorporates its own polymorphic message-passing system. The AppWare Foundation message system is implemented as a hand-crafted layer on top of the native message system. This design allows the developer to gain access to the native message system.

Within the Foundation Series, the Font Component provides support for platform-independent font selection and management. The Character Component provides support for processing character data or text independently of language and character encoding. This is also supported by international extensions in several other software component families. The Foundation Series also supports device management, including support for printers, keyboard control, system control and graphics output to non-screen devices.

The Connectivity Series

The Connectivity Series provides inter- and intra-application connectivity and communication facilities for standalone or mixed computing environments (see Figure 5 on page 9 of the hard copy). The Connectivity Series supports: named pipes and sockets, inter-application messages (datagrams), object linking and embedding (OLE), Apple Edition Manager (P & S) and clipboard management.

The AppWare Foundation supports the creation, opening, querying and closing of named pipes. Pipes provide a file-like interface for streamed communication between tasks. A pipe is created much like a file. Once created, a pipe may be opened by another task to exchange data with the pipe's creator.

Named pipes can be used in a distributed application environment. In this case, a server creates a named pipe as a published service and clients can then access the pipe by name. With the Pipe Network Extension, named pipes can also be accessed by remote clients.

The User Interface Series

The user interface is a crucial part of most applications, since many users base their impressions of an application on how the application

looks and how easy it is to use. From a developer perspective, the user interface typically consumes a large percentage of the engineering resources required for application development (see Figure 6 on page 10 of the hard copy).

Because of its critical nature, much effort has been devoted to developing AppWare Foundation's sophisticated user interface facilities. The User Interface Series provides complete GUI support and manages a superset of operating system interface objects like windows, lists and buttons.

Its facilities include a fully nestable window manager, virtual viewports, multi-font text, embedded graphics for all objects, universal edit-in-place, and a context-sensitive help manager.

The AppWare Foundation currently provides two layers of user interface facilities. For situations where only simple dialogs are required, the Dialog Component provides a set of functions, messages and data structures that support a variety of dialogs. The Dialog Component supports the design and development of most of the modal dialogs found in applications. A special function is provided to easily display message boxes, also called alerts.

For more complex interfaces, the AppWare Foundation provides built-in controls that can be nested. These controls include windows, voids and boxes, which usually contain buttons, boxes, edit text areas, display areas, lists, sliders and tables. A large variety of menu types are also available. To provide more flexibility, a sophisticated GUI data management facility allows the developer to associate any combination of text and graphics inside the controls.

The User Interface Series currently supports the following GUI standards: MS Windows, Apple Macintosh and UNIX Motif, with support for OS/2 standards pending.

Superset Functionality

The goal of all platform portability toolkits is to provide the same functionality on all platforms. This may sound simple, but it’s actually quite difficult. For example, almost all applications allow their users to enter text in one form or another; text-editing ranges from basic editing, such as editable data fields in a communications package, to highly sophisticated word processing. Features available on all platforms, like scrollable views, are no problem. But how should the compatibility package handle features that are not available on all platforms, like multiple fonts in a view, undo capability, text areas greater than 32KB characters.

Platform portability toolkits typically take one of two approaches: either they provide only common features (least-common-denominator toolkits) or they provide all the features (superset functionality). The least-common-denominator approach is very efficient, but meets few functionality requirements. Providing a true superset of the features available on each platform is probably impossible.

The term "superset functionality" is used to describe the AppWare Foundation because those features that can be implemented on every platform are implemented on every platform. AppWare Foundation currently provides a wide range of support for operating system services, GUI services and connectivity in a deep, broad manner.

For example, in the Foundation Series, the File Component provides support for file I/O, file system management (copying, renaming and directory traversal), temporary file support, file aliases, and access to file and directory attributes. The Table Component from the User Interface Series provides support for tables with dividers, the ability to resize the tables, the ability to select multiple cells within the table, the ability to reorder table entries, and the ability to put various types of data into the table cells.

AppWare Foundation also supports many specific features of different GUI environments, including:

- The two types of windows supported by XWindows systems: widgets ("heavyweight" windows) and gadgets ("lightweight" windows)
- Child window functionality
- A method to port RTF hypertext help for MS Windows to all platforms
- Printing support for all platforms

And since the AppWare Foundation is an extensible toolkit, developers can add functionality through either the C or C++ interfaces.

A Scalable Architecture

The goal of a system that provides superset functionality is to enable developers to produce applications that are as efficient and feature-rich as those written with native code. To meet this goal, a toolkit must be developed to utilize the services provided by native operating systems while implementing any missing capabilities using lower-level facilities of the system. The AppWare Foundation offers the most efficient implementation

for developers because it provides multiple implementations based on the capabilities of native operating systems and the requirements of applications.

When a developer decides which features are needed, the AppWare Foundation provides a component appropriate for the situation. For example, a developer needing single-font text editing on MS Windows would not select the Edit Text component that provides multifont editing. Instead, he or she would select a more efficient component, single-font editing component built on top of the MS Windows edit control.

Three AppWare Foundation features support scalability. First, functionality is provided in discrete component modules—developers use only the components needed. Next, the AppWare Foundation provides multiple implementations of functionality in any given component family so a developer can evaluate the choices in each family and select the desired level of functionality. Third, all components within a component family share the same API, so developers don’t have to rewrite calls to implement different sets of features within a component family. As a result of these features, the AppWare Foundation provides the functionality developers need without adversely affecting the size or performance of an application.

Fully Native Implementation

There are two basic approaches to writing a toolkit such as the AppWare Foundation. The first is to develop a virtual abstraction (or toolkit emulation package) that is the same for all platforms; the second is to handcraft the toolkit on all platforms.

The AppWare Foundation was developed using the second approach. The system was handcrafted for each environment, and the different platform implementations share very little code. All implementations are built using native language and tools. And since the AppWare Foundation works alongside native code, a developer can incrementally port an application to the AppWare Foundation to protect the value of existing code. Because of this approach, the AppWare Foundation buys the developer efficiency, native look and feel, and compatibility with native calls.

Efficiency

> Benchmarks of applications based on the AppWare Foundation technology against the most efficient native implementations show consistently identical performance between the two applications — except when the AppWare Foundation implementation is faster. This is because the AppWare Foundation is built using the same efficient techniques native application developers use. AppWare Foundation

improves the native operating systems in weak areas, so the AppWare Foundation implementations are sometimes faster.

Look and Feel

The look and feel of a system developed to the AppWare Foundation is 100% native because the AppWare Foundation is the native GUI — rather than an emulation of the native GUI. As operating system suppliers improve their native toolkits, the AppWare Foundation improves with them. As a result, applications evolve automatically.

Long-Term Compatibility

Native implementations of the AppWare Foundation ensure compatibility with native systems in other ways. For example, the AppWare Foundation can read and write native resources and maintains compatibility with native GUI builders, including Microsoft's Visual C++, Borland's Resource Workshop and Novell's Visual AppBuilder. The AppWare Foundation is also compatible with native message systems, allowing developers to use the native message system when necessary. This means AppWare Foundation applications can interact transparently with non-AppWare Foundation applications. With the evolution of AppWare, the AppWare Foundation will become the common point of entry for network services.

Open System

The AppWare Foundation's open architecture gives developers a great deal of versatility and provides several major benefits that increase programmer productivity.

- AppWare Foundation works with many languages such as C, and C++

- AppWare Foundation is architected for both procedural and object-oriented programming, allowing a development team to start with an API-style interface and then incrementally transition to an object-oriented framework.

- AppWare Foundation integrates with native interface extensions including CDefs, MDefs, LDefs, controls, new classes and new widgets.

- AppWare Foundation integrates with important new technologies. For example, the AppWare Foundation integrates with Adobe Type Manager (ATM) via the back door, allowing the AppWare Foundation to provide features such as text rotation through ATM (or TrueType).

- Developers are free to choose their platform, compiler, linker and debugger. The AppWare Foundation works with Symantec, Borland, Microsoft, MPW, Lightspeed, SABER, and GNU compilers, as well as with Multiscope, Codeview, and SADE debuggers.

AppWare Foundation provides a stable base for developers of high-level application tools such as frameworks, class libraries, 4th-generation languages and visual programming tools. Tool suppliers can focus on productivity issues instead of platform portability problems.

Its open architecture also allows developers to integrate the AppWare Foundation with a variety of third-party tools. This provides developers with more functionality, including PostScript, online help engines, Visual Basic, Borland's Resource Workshop and Microsoft's Dialog Editor.

A Layered Architecture

The AppWare Foundation provides two layers built on top of the native operating system as shown in Figure 7 (see page 14 of the hard copy). AppWare Foundation Object Classes are a C++ class library implemented with the AppWare Foundation Libraries. The benefits of this layered architecture include:

- **Ability to Extend.** AppWare Foundation Object Classes are easy to extend. Rather than learning the intricacies of each platform, developers using the AppWare Foundation Object Classes can review the portable AppWare Foundation C source code rather than deal with the native code when deriving new classes. This allows developers to create a subclass and supply new virtual methods for the desired behaviors.

- **Facilitated migration from C to C++.** AppWare Foundation Object Classes integrate well with both AppWare Foundation objects and native C code. Since AppWare Foundation Object Classes are built on top of the AppWare Foundation Libraries, AppWare Foundation Object Classes objects can be converted to and from AppWare Foundation objects. C code can easily take advantage of new AppWare Foundation Object Classes-based C++ classes and still compile under any ANSI C compiler.

- **Power of C++.** AppWare Foundation Object Classes provide a robust, object-oriented class library for C++ developers. A combination of class derivation and instantiation encourages code reuse.

This architecture provides a development team with a maximum

amount of flexibility in an ongoing development effort. What's more, the flexibility is available on all development platforms.

Summary

The real proof of a toolkit's value is whether users successfully build commercial-quality applications with it. Developers have used the different technologies found in the AppWare Foundation to successfully implement more than a million lines of code in real-world applications. This success is directly related to focusing on the needs of developers, not on the limitations of operating systems. This developer-oriented approach ensures an application-driven product and Novell's commitment to work toward improving developer productivity.

As AppWare continues to mature, the AppWare Foundation will play an important role in providing a portability platform that allows distributed applications to directly access the features and functions of Novell's network operating systems.