The Design of ManageWORKS: A User Interface Framework

by Dennis G. Giokas and John C. Rokicki

ABSTRACT

The ManageWORKS Workgroup Administrator for Windows software product is Digital's integration platform for system and network management of heterogeneous local area networks. The ManageWORKS product enables multiple, heterogeneous network operating system and network interconnect device management from a single PC running under the Microsoft Windows operating system. The ManageWORKS software is a user interface framework; that is, the services it provides are primarily targeted at the integration of the user interface elements of management applications. It manifests the organizational, navigational, and functional elements of system and network management in a coherent whole. Viewers, such as the hierarchical outline viewer and the topological relationships viewer that are components of the ManageWORKS software, provide the organizational and navigational elements of the system. Management applications developed by Digital and by third parties through the ManageWORKS Software Developer's Kit provide the functional elements to manage network entities. This paper discusses the user interface design that implements these three elements and the software system design that supports the user interface framework.

INTRODUCTION

The ManageWORKS Workgroup Administrator for Windows software product is Digital's strategic tool for providing system and network management of heterogeneous local area networks (LANs). It serves as Digital's platform for the integration of PC LAN management. From the perspective of the end user, i.e., the LAN system administrator and network manager, the ManageWORKS product comprises a suite of modules that integrates a diverse set of management activities into one workspace. From the perspective of the developer of system and network management applications, the ManageWORKS product is an extensible and flexible software framework for the rapid development of integrated management modules, all of which presents a consistent user interface.

The design of the management system was user centric, i.e., usability was the top priority. Thus, we began the design work without any preconceived notions about the management software system design. The design that emerged and that is documented in this paper was driven solely by the user interface paradigm developed and tested with our customers.

This paper focuses on how the ManageWORKS software presents and

integrates its functionality to the end user. Specifically, the paper presents details of the user interface paradigm and discusses the design rationale and the design methods employed. The paper also discusses the design of ManageWORKS software in support of the user interface framework.

DRIVING FORCES BEHIND THE DESIGN

The ManageWORKS software was originally released as a component of the PATHWORKS version 5.0 for DOS and Windows product. The foci for that PATHWORKS release set the tone for the ManageWORKS design. The PATHWORKS version 5.0 design objectives were to

1. Enhance the usability of the PATHWORKS product. Since the PATHWORKS system was rooted in a command line-based user interface, the goal was to develop a graphical user interface for the system that was based on the Microsoft Windows operating system. Such a user interface would be contemporary, easier to learn, and easier to use.

2. Enhance the manageability of the PATHWORKS product. The goal was to reduce the cost of ownership by improving the installation, configuration, and administration of the system.

The ManageWORKS design team used two voice-of-the-customer techniques to provide more depth and detail for the two high-level product design objectives. First, the team used Contextual Inquiry to determine a customer profile and to develop a clearer statement of the user's work.[1] Then, the team tested user interface prototypes with customers by means of formal usability testing. From 15 to 20 customers and users participated in each of three rounds of usability testing.

Early in the investigation, Contextual Inquiry revealed that the profile of the PATHWORKS system administrator had changed drastically during the five years since the PATHWORKS product was first released. A typical system administrator in the era of PATHWORKS version 1.0 had been a VAX/VMS system manager who inherited the responsibility of installing and managing a PC file and print-sharing product. The interface into the system was a VT-class terminal running command line-based utilities. Today, a system administrator is usually a PC user who is quite familiar with graphical user interfaces. Such an administrator is more likely to be trained in the installation, configuration, and management of PCs and PC networking software than his/her predecessors. This change in the profile encouraged us to shift the PATHWORKS focus from using host-based command line utilities to manage the system to using client-based graphical utilities.

We also profiled the customer network configuration. During the same five years, it changed from a very simple and homogeneous environment with just a few PATHWORKS servers to a

medium-to-large heterogeneous PC LAN. At present, configurations comprise network operating systems that consist of Novell NetWare, Microsoft LAN Manager, and Apple AppleShare file and print services, as well as other services that are emerging in the PC LAN environment. The network operating systems are deployed on their native platforms and by Digital on the OpenVMS and DEC OSF/1 platforms. Each system has its own tools to manage the clients and the servers. Each has a different user interface that results in a long learning curve and thus high training costs or low productivity for system administrators. Customers reported that they desired tools with a consistent user interface to manage this diversity.

The team employed software usability testing throughout the development life cycle. Two usability tests were performed with early design prototypes; the final test was performed with our first pass at a detailed concept design. We performed the usability testing with customers to test user interface and functional element design concepts that we developed as a result of the Contextual Inquiry. The user thus served as a design participant. With each iteration of the formal testing, we tested specific functional concepts in three key areas: (1) mechanisms to navigate among the managed entities, (2) mechanisms to organize these entities, and (3) the functional capability inherent in the management directives supported. (Note that, in this paper, the servers, services, and resources managed by means of the ManageWORKS software are collectively referred to as managed entities.) The major lessons that we learned from this testing effort and then applied to the user interface and software designs are as follows:

1. The ManageWORKS software had to provide mechanisms to navigate among a diverse set of managed entities on the LAN or in some user-defined management domain. Users want to be able to view and thus "discover" the entities that are to be managed. The system had to present the managed entities in graphical display formats that were familiar and enticing to users. Users welcome the ability to support different styles of presentation. Finally, users need easy mechanisms to navigate through the hierarchy of an entity.

2. Navigation mechanisms, as just described, work well for novice users but become tedious and constraining for more experienced users, as we could attest to after our experience with the prototypes. The solution that we presented to users allowed them to create custom views of their managed entities, i.e., to organize their management domains. This concept was well received by users during usability testing.

3. The ManageWORKS product had to provide mechanisms that consistently performed the functions that were common among a diverse set of management applications. The

product design presents users with an object-oriented view of the managed environment. The building block of this design is the object, an abstraction of a manageable entity such as a server or a network router. Each object is a member of a single object class that describes the set of object instances within it. The ManageWORKS application renders objects to the user as icons in a viewer. For example, for a LAN that contains three NetWare servers, the object class called NetWare Servers would contain three objects, each of which represents one of the three individual NetWare servers on the LAN. When users focus on an object, the tool reveals which actions are valid in the object's current context. This approach differs from the traditional command line approach in which the user first selects the utility (action) and then specifies the objects upon which to act. Interestingly, whereas novice users found this object-focused concept easy to grasp, those who considered themselves strong users of the traditional command line management utilities experienced difficulty in grasping the new concept.

4. The typical customer has a diverse and large (200 to 1,000) number of entities to manage. To address this need, the prototype testing presented users with the ability to manage more than one entity at the same time and the ability to manage many entities as one. Users liked being able to view and modify the properties of multiple entities at the same time as well as being able to modify the same property across a set of like entities.

5. In addition to providing a consistent user interface, the ManageWORKS product should integrate the management tools into one workspace. User feedback led to the design of the user interface framework as the delivery vehicle for a diverse set of management applications.

THE KEY SOFTWARE DESIGN PRINCIPLES

At this point in the development cycle, the design focus shifted from developing and testing user interface and functionality concepts to designing the ManageWORKS software itself. With what we considered to be a good understanding of the user's needs, we proceeded to design a software architecture to support those requirements.

Prior architectures that were familiar to the design team served as starting points for the design. The following two examples represent sources of design concepts that we employed and adapted to suit our objectives. Each represents an opposing end of the spectrum with respect to design objectives and implementation.

The ManageWORKS team adopted the concept of plug-in modules, a software design that is supported by the Windows Dynamic Link Library (DLL) architecture.[2] The design is also in common use by many Windows applications including the Windows Control Panel, the utility that manages the local desktop's configuration and user preferences.[3]

The next challenge was to decide how much constraint to impose on the design of the ManageWORKS' plug-in modules and how consistent the modules must be. Digital's extensible enterprise management director, the DECmcc product, incorporated some excellent concepts.[4] In particular, our design was influenced by the way in which DECmcc layered the management responsibility into presentation modules, functional modules, and access modules. Early in the design process, we decided to separate the navigation and presentation of managed entities from the access and functional management of the entities.

Another DECmcc concept, which is used, for example, in the access module layer, was the presentation of a consistent view to the layers above.[4] This concept, however, was not suitable for the ManageWORKS design because it would have placed constraints on the user interface design, in particular, on the presentation of the attributes of managed entities. The design team was not willing to compromise on this aspect of the design.

Thus, we decided on a ManageWORKS design that can best be described as a user interface framework. The initial release, which was a component of PATHWORKS version 5.0 for DOS and Windows, offered few services other than to tie together the user interface elements required for system and network management. The user interface services needed were dictated by the five user interface requirements previously described.

The ManageWORKS design incorporates two types of plug-in modules: navigation modules, referred to in the ManageWORKS product as Object Navigation Modules (ONMs), and application modules, referred to as Object Management Modules (OMMs). The ManageWORKS framework controls the control flow and messaging between the modules.

ONMs allow for any number of navigation models to be supported and used singly or simultaneously by the user. Although, by design, ONMs possess no knowledge of the managed entities or entity relationships they display, they do possess the ability to display entities with the relationships inherent in them. ONMs also provide the mechanisms for browsing and navigating through the management hierarchy. In addition to navigation capabilities, ONMs provide the user interface for organizing entities into a user-defined management domain.

The OMMs are responsible for managing the entities. The OMM design has three key components.

1. OMMs provide the methods used to manage the entities. These methods include the functions of discover, create, view, modify, and delete. The OMMs also have the option of presenting to the user additional methods. That is, since each OMM knows how to manage the entities for which it is responsible, it knows which actions can be applied to an entity based on the entity's current state and the user's context.

2. OMMs provide access to the managed entities. An OMM can use any interprocess communication mechanism to access or to manage an entity. Examples include the task-to-task, remote procedure call, and object request broker mechanisms. Since a PC LAN environment affords no common way for a management director to communicate with all the types of devices present, the design team decided to leave the choice of access mechanism up to the OMM.

3. OMMs provide the user interfaces required for managing the entities. This design component allows developers to present an interface that best suits the needs of the user and best maps to the entity being managed. It also allows for flexibility, evolution, and innovation in the user interface of OMMs. The ManageWORKS design team did not want to impose a user interface style or present a user interface that was compromised by the diversity of applications that we envisioned running within the context of the framework, e.g., by being the least common denominator. Even though one of the key product design goals was a consistent user interface, we felt that it was important to allow the OMMs to control the user interfaces. First, we thought the design benefits outweighed the risk of any inconsistency. Second, we encouraged, but did not enforce, consistency by means of a user interface style guide and common libraries that implemented those guidelines.[5,6]

The plug-in modules also have a residual benefit. Because these modules can easily be added to or removed from the environment, they provide an easy way to extend and to customize the ManageWORKS product. Digital and third parties can develop new ONMs and OMMs and simply enroll them into the system. Users have the additional benefit of being able to customize the product to support only the ONMs and OMMs that are useful in their environment.

THE USER INTERFACE OF ONMs AND OMMs

Given the key software design elements presented in the previous section, the focus of the paper now returns to the user interface. This section describes what was implemented to support the customer requirements and the design framework.

The user interface framework manifests the organizational, navigational, and functional elements of system and network management in a coherent whole. For example, the first three menus on the ManageWORKS menu bar -- Viewer, Edit Viewer, and Actions -- are all the tools the user needs to manage entities. A discussion of the Viewer and Edit Viewer menus follows.

By means of the ManageWORKS Viewer menu, ONMs present display elements, called viewers, to the user. Each instance of a window that an ONM creates is considered a viewer. A ManageWORKS viewer is one of the organizational elements for the user and is the root-level object for navigation. Each viewer is a viewport into a set of managed entities that the user may be browsing and navigating through. A viewer is analogous to a word processor's document, i.e., a viewer is a ManageWORKS "document." Just as you can create new documents and open, close, or edit existing documents when you use a word processing application, you can perform the same functions on viewers when using the ManageWORKS software.

ManageWORKS ONMs are responsible for the navigational and organizational display properties. The current ManageWORKS release comes with two ONMs. One ONM supports a hierarchical display of managed entities. This display is rendered in a single viewer window graphically as a tree or textually as an outline. The other available ONM supports the relational display of managed entities, rendered as a map. The map ONM can also support a hierarchy; each map is rendered in a new viewer instance. Figure 1 shows ManageWORKS with two hierarchical viewer styles and a map viewer. The hierarchical views are the Outline view (shown in the Browser viewer) and the Outline Tree view (shown in the IP Hierarchical View viewer). In addition to the map viewer (shown in the IP Discovery viewer), note the navigation window for the map viewer (shown in the IP Discovery (Navigator) viewer). This view shows a scaled map; the entire contents of the map viewer appears in a rectangular outline, which represents the user's current viewport into the data. The user can use the PC pointing device to drag and reposition the viewport.

[Figure 1 (ManageWORKS Viewers) is not available in ASCII format.]

Because the ONM maintains context when the user "edits," i.e., modifies, the contents of a viewer, the user can customize or organize the managed entities as desired. By means of the Edit Viewer, ONMs allow user customization within a viewer with the support of user-definable hierarchies. For example, each instance of a viewer can represent a different management domain for the user. The benefit is that the user can find objects and then arrange them into hierarchies that are most useful.

As stated earlier, OMMs control the user interfaces for displaying and modifying managed entity properties. The ManageWORKS framework provides for consistency in how the OMMs

invoke the user interfaces and in how the user interfaces interrelate to the ONMs.

The consistency starts with the ManageWORKS Actions menu. The basic management directives on managed entities originate from this menu. The major challenge in designing this menu was to avoid using too many menu items, menu items that would change constantly (i.e., by addition or deletion), menu items that had three or four levels of hierarchy, and menu items that were not context sensitive to what the user was doing. The objective was to find a small set of words that conveyed the management functions the user would most often perform. We felt that these words should always be present in the Actions menu, but to eliminate confusion for the user, they should be rendered inactive when not valid. On the other hand, we realized that this small set of menu choices could never fully support the actions on managed entities; therefore, the software had to provide an extensibility mechanism.

We began the design process by developing an entity/action matrix. One axis contained a list of the entities that we envisioned being managed from the ManageWORKS software. The other axis contained a list of the actions that could be performed on the entities. We marked the intersections of the axes. In forming the list of actions, we chose words that were used in existing products that managed the same entities, words that we thought should be considered in a good user interface, and finally, synonyms to those words already listed. This approach gave us a clear picture of the common actions and also provided a thesaurus of words from which to choose. The common actions on managed entities that emerged from this exercise were

1. Make a new entity of some type.

2. Display all the managed entities.

3. View and modify the entity's properties.

4. Eliminate the entity.

The ManageWORKS software supports these common actions through the following Action menu choices:

1. Create. Choose Create to make a new entity.

2. Expand. Choose Expand to view all the entities that the ManageWORKS software is managing.

3. Properties. Choose Properties to display a dialog box that manifests all the entity's properties. The user can then view the properties and make modifications, as appropriate.

4. Delete. Choose Delete to eliminate the entity.

The design of the Properties dialog box is one of the key user interface style elements of the ManageWORKS product; however, ManageWORKS does not enforce or provide for this element. Rather, the consistency is a function of a user interface style guide for OMMs and some common library routines that support this user interface style.[5,6] Figure 2 shows the dialog boxes of two of the three OMMs that come with the current ManageWORKS product: the Simple Network Management Protocol (SNMP) Manager OMM and the LAN Manager (LM) server management OMM. (The third OMM, for NetWare servers, is not shown.) Note the Selected Objects field in the SNMP dialog box. The ManageWORKS software allows the user to select multiple objects of the same class from a viewer and to invoke an OMM method. The list of selected objects is contained within this drop-down list box. The user can easily view the attributes of different objects from the same dialog box. The dialog box displays various sets of managed entity properties. The user can select the desired set of properties from the View or Modify drop-down list boxes.

[Figure 2 (ManageWORKS OMM Properties Dialog Boxes) is not available in ASCII format.]

Figure 2 demonstrates that two dialog boxes can be active at the same time. This feature supports the ManageWORKS design requirement that the user be able to manage more than one entity at a time. The ManageWORKS product supports, in effect, threads of execution to allow multiple OMMs to be active simultaneously. Support for the design principle of managing many entities as easily as one is not a function of the ManageWORKS software but of the OMMs, since OMMs control the methods used to manage entities.

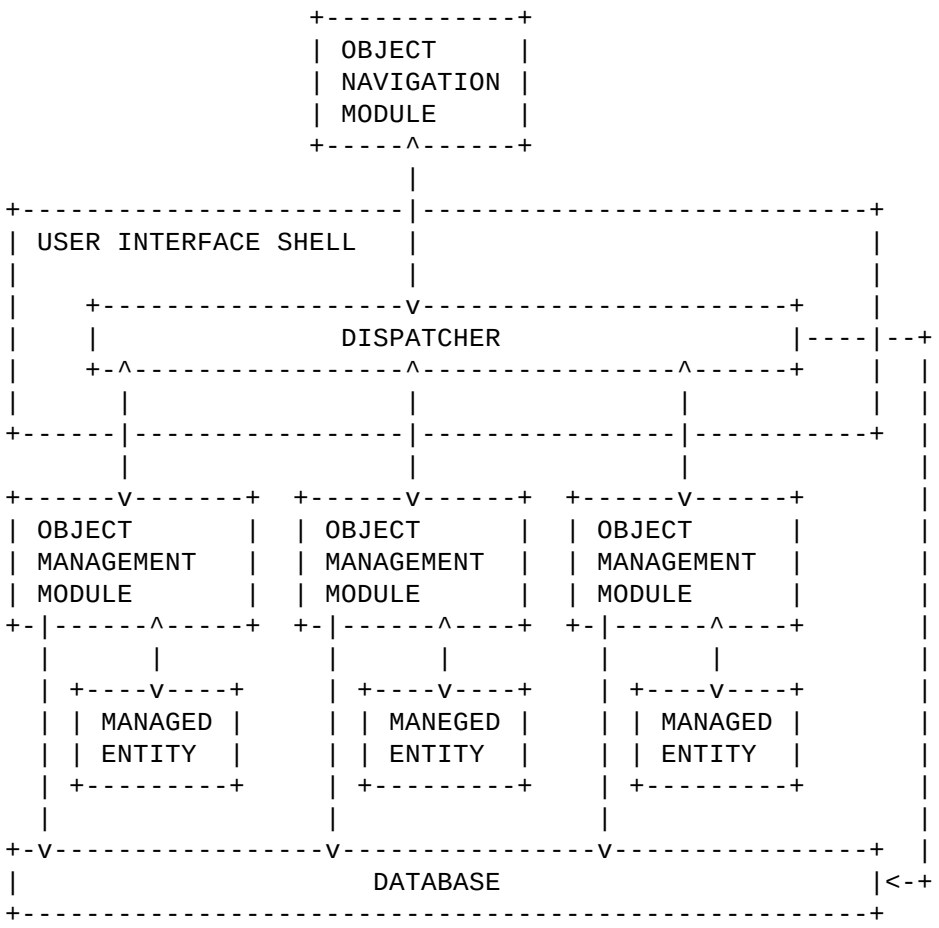
THE SOFTWARE SYSTEM DESIGN OF ManageWORKS

The focus of the paper now shifts to the ManageWORKS internals that support the design principles and user interface just described.

The Application Framework

As an application, the ManageWORKS product is merely a software framework for integrating its top-level user interface with the user interfaces of the OMMs and ONMs. The ManageWORKS application consists of two main components: (1) the user interface shell and (2) the dispatcher. Figure 3 depicts the relationship between these ManageWORKS components and the OMMs and ONMs.

Figure 3 ManageWORKS Application Architecture



The user interface shell is a standard Microsoft Windows application that supports the top-level Windows user interface components -- the main application window and its menu bar, tool ribbon, and status bar. The user interface shell translates all user interaction by means of the menus, tool ribbon, and mouse actions into OMM and ONM application programming interfaces (APIs) to perform work for the end user. The shell is also responsible for initializing and terminating the application, including the dispatcher.

The dispatcher is responsible for maintaining a link between the user interface shell and all the OMMs, as well as for providing service routines. The dispatcher loads and initializes all OMMs present based on an initialization file that the end user configures at installation time (or, if subsequent modules are added, by means of the Management Module Setup program). To enable this routing to occur, the dispatcher maintains a list of all OMMs loaded and the object classes that they support.

One service that the dispatcher provides for OMMs and ONMs is the ability to modify the menu bar. OMMs and ONMs may add and set menu items but only through the APIs. The ManageWORKS software ultimately controls what gets displayed in the menus based on what objects are selected in a viewer, which prevents the modules from directly manipulating the menu bar.

The Application Programming Interfaces

Once we had defined the concepts of the ManageWORKS user interface and object classes, we designed a common set of APIs that all OMM and ONM developers would employ. The APIs that emerged focused primarily on the object -- both its class and its instance. Because the current set of object-oriented languages and tools does not map well to the services supplied by the Windows system, these APIs are in a more conventional C/Pascal programming language style rather than in a C++ style.

The APIs that an OMM must support fall into three categories based upon their scope of operation: (1) module based, (2) class based, and (3) object based. All APIs have parameters that contain information pertinent to the API call, including the object identifier (OID), which identifies the object on which to perform the operation.

Module-based APIs perform initialization, termination, and information reporting for the entire OMM. The initialization includes determining how many object classes an OMM supports. This function is important because an OMM can support more than one class, e.g., a hierarchy of classes. By checking for software dependencies on the operating system or support libraries, the OMM can also make sure that the computer environment is capable of supporting the OMM. For example, Digital's implementation of the OMM that manages NetWare servers requires that the NetWare client be installed and configured on the PC. Module termination occurs before the ManageWORKS software terminates, which allows OMMs to clean up any resources they may have used. The information function provides information such as the module's name and copyright information.

Class-based APIs support the actions that apply to all objects within a class. These functions include initialization, termination, configuration, and reporting information about what actions and properties can be accessed by the end user in the ManageWORKS user interface. A class-based configuration API presents a configuration window for each class to the user; the user can then change the behavior of the object class. For example, the user can indicate whether or not files on a disk with hidden or system attributes or hidden LAN Manager file services should be displayed.

Object-based APIs provide the ability to manipulate individual objects within the ManageWORKS software. With these APIs, OMMs

can accomplish all the base actions and those operations provided for in the user interface. These APIs include functions to create, delete, insert, remove, copy, get and set properties, display a properties dialog box, maintain containership relationships (e.g., technology-based hierarchies), and maintain classes that can be created and inserted into an object. Approximately 30 APIs (a small manageable set) must be implemented to be ManageWORKS compliant.

Each class- or object-based API requires an OID or list of OIDs on which to perform the operation. When called, each class API acts on a single object class. The caller manages all memory needed for the successful completion of an API, i.e., no API returns a pointer to data. APIs that can return a variable amount of information use a two-step calling convention. The first call determines the buffer size required to hold all the data; the second call retrieves that data. This two-call approach requires OMMs to efficiently gather information using OMM-specific information caches to store information retrieved from the managed entity.

ONMs contain all the module-, class-, and object-based APIs that exist in a standard OMM but also contain some viewer-specific APIs. These APIs include functions to display viewers, select displayed objects, expand objects, update objects, and retrieve displayed objects. New ONMs can be developed using these APIs.

The Object Identifier

To represent objects within the ManageWORKS software, we chose the approach of assigning an OID to each object in the system. This number embodies the information of the class to which the object belongs as well as the uniqueness of the individual instance of an object within the class.

The assignment of an OID to an object is the responsibility of the OMM. The ManageWORKS software dynamically assigns to an object class an OID that represents the class, and the OMM is responsible for creating the unique instance values within the context of that class. This approach allows OMMs the flexibility of using any strategy to assign these values, e.g., sequential assignment or mapping to a particular technology, such as an external database record.

Each OID is a 32-bit number; the high 12 bits contain information that identifies the class to which the object belongs to. This bit arrangement places a limit, (2**12)-1, i.e., 4095 (a value of 0 is invalid), on the number of classes that can be active with ManageWORKS at any one time. The low 20 bits provide the uniqueness for each object instance within the class, providing for up to (2**20)-1, i.e., more than 1 million, individual instances within a single class. The advantages to using an OID lie in allowing objects to store information in any format they

wish and using access functions to get at that information in a consistent manner.

Storing Information about Objects

Although the OMMs are responsible for assigning OIDs to objects within a class and for storing information about each object that can be managed, we did not want every OMM under development to have to create its own mechanism to accomplish these tasks. We decided to create an object database that would store information about objects and generate new OIDs for the OMMs.

Initial designs of this object database were to support multiple users and thus allow the sharing of information between multiple ManageWORKS users and other applications. Because the schedule for the first release of the ManageWORKS software did not give us ample time to employ a commercially available database, we decided to create our own database to support the management of object classes and object instances. This database supports only a single user and consists of indexed files for (1) object information, (2) class information, and (3) containership information. The existence of these files is hidden under a database API, which supports all the management aspects of objects, from creating and deleting classes and objects to reading and modifying attributes of those objects.

To allow future changes in the underlying technology of the database, we placed the database code into a DLL. For the second release, we created a new database DLL, with the same APIs, that works with Borland's dBase IV database implementation. By simply replacing the database DLL, all OMMs can now take advantage of having information shared between ManageWORKS users across the network. This design allows for comanagement of the LAN by multiple network administrators who have the same information available. The OMMs do not have to make any source code changes to work with this new database DLL, but additional APIs are present to allow for the use of advanced database features.

Before an OMM can create objects in the database, the object class itself must be created in the database. Because it dynamically assigns OIDs, the object database must store unique information about the class along with the OID. Each OMM must register an object class, where each class has a name that can be presented to the user in the user interface, and a class tag. The class tag is a 64-byte character string that must be unique among all OMMs. The database dynamically assigns an OID to a newly created class and maintains that mapping to the class tag. We decided that using a unique 64-byte character string would result in less conflict among OMM developers than assigning hard-coded OID values to each customer that wanted to develop an OMM. By not hard-coding the values, we ensured that each newly created object class would receive the next OID value. Thus, different end users who are using different sets of OMMs may have different OID

values assigned to each of the object classes.

OMMs can use this object database to create object classes or objects within those classes, and to store any amount of information with each object. Most objects store enough information to get to another data source, thereby preventing information in the database from becoming inconsistent with the managed entity. For example, a NetWare Server OMM saves only the server name in the database because with that name the OMM can make NetWare API calls to retrieve other information.

When the object database creates an object, it assigns the object an OID within the space of that object class. Thus, OMMs can rely on the database for creating unique OIDs for each object in the system.

Another feature of the object database is the concept of transient and permanent objects. The object database DLL writes transient objects not into the database files but rather to global system memory in the Windows operating system. Having the objects in memory creates a large performance gain and avoids the problems associated with disk thrashing. To indicate the type of object that is created, the object database reserves bit 19 of the OID to use as a flag. If the bit is set by the OMM or ONM, the object is transient. When an object is created in the database, the OID for the class is passed to the database DLL with or without bit 19 set, thus determining whether the object is transient or permanent.

In our initial development work, we quickly discovered that creating all the OID entries in a database file diminished performance. This problem was most evident in the development of the DOS file system OMM. This OMM enumerates directories, which causes a disk seek operation and a disk read operation for the enumeration. Next a write of the object to the database file on the same disk causes another disk seek/write operation. This resulted in tremendous disk thrashing. We envisioned that many OMMs would enumerate and create a list of contained objects each time an object is expanded, so we wanted this operation to be fast and efficient.

Introducing New OMMs and ONMs into the ManageWORKS Software

In traditional software development, the addition of new functionality into an application generally requires source code modification and recompilation. Clearly, this approach would not allow ManageWORKS developers to meet the goal of providing an extensible application framework. Developers needed a way to write software that could become part of the ManageWORKS application without requiring changes to the application.

Since the ManageWORKS software runs in the Microsoft Windows operating system environment, software developers were able to

take advantage of many features of the Windows system. We used DLLs to provide an extensible framework for the ManageWORKS product.

By creating a DLL that conforms to the set of APIs needed to manage an object or to implement a viewer, we can add new DLLs at any time to add functionality to the ManageWORKS software. Therefore, all OMMs and ONMs must be implemented as DLLs. The registration process needed to be simple and dynamic for these DLLs. Using a Windows application initialization (INI) file, the dispatcher reads the list of entries in the file and attempts to load and initialize all OMMs and ONMs defined. End users can add new OMMs by running the ManageWORKS Management Module Setup program, which simplifies the installation of any OMMs provided by either Digital or a third-party vendor.

When a new OMM is introduced, the ManageWORKS software needs to assign an OID to each object class that the OMM handles. This is accomplished by asking the dispatcher for an OID for the class based upon a supplied class tag. The dispatcher then uses the object database to have the OID assigned. The dispatcher's use of the object database ensures that the OID for the class is unique to that class. OMMs can ask the object database directly, but this is merely a side effect of the dispatcher's use of the object database and is not recommended.

Interactions between ManageWORKS Components

Most ManageWORKS events occur when the user interacts with the user interface, although OMMs and ONMs can generate events that cause communication to occur between the components of the system. The usual flow of control through the ManageWORKS software begins with a viewer, the set of selected objects in a viewer, and the valid managed entity actions in the Action menu. The application uses the dispatcher to call a particular API to the correct OMM for the class of object being operated upon. In this section, we walk through three typical user interaction scenarios. For each scenario, we describe key elements of control flow between the user interface shell, the dispatcher, the ONM involved, and the OMM involved. These scenarios illustrate how the ManageWORKS elements fit and work together to achieve our primary objective, i.e., to design a user interface framework with consistent mechanisms to display, organize, and navigate through management entities for the purpose of managing one or more of those entities.

Scenario 1. This scenario outlines the process of displaying the properties dialog box of the selected object(s) in a viewer.

1. The user has selected one or more objects of the same class in a viewer by clicking with the mouse.

2. The user then chooses the Properties menu item from the Actions menu. As a reminder, this action invokes the properties dialog box, which by style guide convention, supports the viewing and modification of a managed entity's properties.

3. The ManageWORKS software queries the selected viewer for the list of selected objects and obtains the OIDs of the objects from the viewer.

4. The ManageWORKS dispatcher decodes the object class portion of the OID.

5. The ManageWORKS software tells the OMM of that object class to display the properties dialog box for the list of objects (OIDs) supplied.

6. The OMM displays a properties dialog box that contains all the supplied objects. The OMM has complete control of the user interface for this window and complete control over the access to the managed entity mechanism to get and set the properties from the managed entities.

Scenario 2. This scenario outlines the process of expanding a selected set of objects in a hierarchical viewer. Expanding an object results in the display of the object's descendants within the hierarchy defined by the OMM. The user may render this display in a hierarchical fashion with one of the hierarchical view styles or as a descendant portion of a topological view.

1. The user has selected one or more objects in a viewer by clicking with the mouse. The objects may be of the same class or of different classes.

2. The user then chooses the Expand menu item from the Actions menu.

3. The ManageWORKS software queries the selected viewer for the list of selected objects and obtains the OIDs of the objects from the viewer.

4. The ManageWORKS software tells the selected viewer to expand the list of objects supplied (the selected objects from the last call).

5. For each selected object to be expanded, the viewer queries the object by means of the dispatcher for the list of contained objects within that object. The dispatcher calls the OMM that supports the object to get the list of contained objects. The viewer repeats this process for all OIDs to be expanded.

6. For a hierarchical view, the viewer places the list of

objects into the viewer in a hierarchical fashion. For a topological map view, the viewer either creates a new window or replaces the current window, depending on the choice the user has indicated through the customization dialog box. The window shows the descendant set of objects with their topological relationships.

7. For each of the contained objects, the viewer queries the object's OMM by means of the dispatcher for its name and bitmap, and to determine whether it can potentially be expanded by the user. The viewer repeats this process for each contained object to be displayed and then renders each item.

Scenario 3. This scenario outlines the process of dragging and dropping an object onto another object in a viewer. The OMM of the target object controls the semantics of this operation.

1. The viewer controls the drag-and-drop operations.

2. The viewer determines the OIDs of the object(s) that the user is dragging.

3. As the user moves the mouse, the viewer receives mouse move messages from the Windows system and determines if the mouse is over a viewer. The window messages are sent directly to the viewer window.

4. If it is over a viewer, the mouse tells the target viewer what objects the user is dragging over it. The source ONM sends a ManageWORKS-defined Windows message to the target viewer window with the list of OIDs being dragged.

5. The target viewer determines what object the mouse is over and if that object is selected. The set of objects targeted to receive the dropped object comprises either the individual object, or if selected, all the selected objects in the viewer.

6. The target viewer queries the OMM of each target object about what class of object can be dropped on it. If all the target objects can accept the dragged objects, the cursor changes shape to reflect a potentially successful drop. Otherwise, the cursor changes to reflect that the drop would not succeed at this mouse location.

7. When the user drops the objects, the same verification occurs as during the drag operation. If the drop is not going to be successful, the viewer that initiated the drag operation returns the mouse cursor to the original location.

8. If the drop operation passes the verification step, each

object that the user is dragging is copied by the OMM to each target object. This is done iteratively for each dragged object, and each copy has the potential for failure. For example, a DOS file can be dragged to a DOS disk class object, but when the copy is attempted, the disk may not have enough free space to successfully copy the file. When each dragged object is copied, the OMM of the target object is told that it should now contain the new object. This causes the hierarchy to be properly updated. A drag-and-drop operation that is intended to move an object is implemented as a copy followed by a removal of the original.

CONCLUSIONS

We feel that we have been successful at building a unique user interface framework that integrates a diverse set of applications; the design essentially meets all but one of the objectives we established. Because by design we limited the scope of services provided by the framework, we could not meet all of our end-user objectives. Specifically, the responsibility of allowing the user to manage many entities as though they were one fell on the OMMs and not on the framework itself. Although we would have liked the framework to provide this service, such a design was not feasible, given that the OMM controlled both the access to the managed entity and the user interface to view and modify entity properties.

The reader should observe that the first two major releases of the ManageWORKS software provide few core services. The core services include the user interface shell, the viewers, and the object database that ship with the ManageWORKS product and the ManageWORKS Software Developer's Kit. These components serve as a unifying framework for the functional modules, which provide the user with tools to manage entities and are thus the "heart and soul" of the environment. Future development of core framework services is under consideration. Among the areas under active consideration are Windows Object Linking and Embedding (OLE) support and scripting support for inter- and intra-OMM control. Such services would make ONMs and OMMs more consistent, useful, and powerful for the end user. At the same time, these services would free the individual developer from writing this code and thus provide the developer the freedom to focus on the value-added functionality.

ACKNOWLEDGMENTS

Many people contributed a great deal to the design and implementation of the ManageWORKS product. Although the contributors are too numerous to mention individually, we would like to acknowledge the functional groups within the PATHWORKS organization to which they belong, namely, Business Management,

Marketing, Human Factors Engineering, Systems Quality Engineering, Documentation, Release Engineering, Field Test Administration, and, of course, Software Development Engineering.

REFERENCES

1. K. Holtzblat and S. Jones, "Contextual Inquiry: A Participatory Technique for System Design" in Participatory Design: Principles and Practice, A. Namioka and D. Schuler, eds. (Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., 1993).

2. Microsoft Windows Guide to Programming (Redmond, WA: Microsoft Press, 1990).

3. Windows 3.1 Software Developer's Kit, Control Panel Applets in Online Help (Redmond, WA: Microsoft Press, 1992).

4. C. Strutt, and D. Shurtleff, "Architecture for an Integrated, Extensible Enterprise Management Director" in Integrated Network Management, vol. 1, B. Meandzja and J. Westcott, eds. (Amsterdam: North-Holland, Elsevier, 1989): 61-72.

5. ManageWORKS Programming Guide (Maynard, MA: Digital Equipment Corporation, Order No. AA-QADFB-TE, 1994).

6. ManageWORKS Programmer's Reference (Maynard, MA: Digital Equipment Corporation, Order No. AA-QADGB-TE, 1994).

BIOGRAPHIES

Dennis G. Giokas Dennis Giokas is currently a senior associate with Symmetrix, Inc. While at Digital from 1984 to 1995, he was a consulting engineer in the PATHWORKS group. He co-led PATHWORKS V5.0 and architected the user interface and system management tools. He was also architect and manager for the PC DECwindows program. Previously, Dennis worked at Arco Oil & Gas and The Foxboro Company developing process control software. He holds a Bachelor of Music from the University of Massachusetts at Lowell, a Master of Music from the New England Conservatory, and a M.S.C.S. from Boston University.

John C. Rokicki John Rokicki, the project leader for ManageWORKS Workgroup Administrator, is a principal software engineer within Digital's Network Operating Systems engineering organization. His primary responsibility is the design and implementation of the base services of the ManageWORKS product. Before joining Digital in 1990, he was employed by Data General Corp. and Sytron Inc. John holds a B.S. (1989) in computer science from Worcester Polytechnic Institute.

TRADEMARKS

The following are trademarks of Digital Equipment Corporation: DEC, DEC OSF/1, DECmcc, Digital, ManageWORKS, OpenVMS, PATHWORKS, VAX, and VMS.

Apple and AppleShare are registered trademarks of Apple Computer, Inc.

dBase IV is a registered trademark of Borland International, Inc.

Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation.

NetWare and Novell are registered trademarks of Novell, Inc.

OSF/1 is a registered trademark of the Open Software Foundation, Inc.