

5 Network Software

Introduction

In this chapter we discuss network systems. The specific systems selected for discussion here were chosen because they made the best possible illustration of the variety of system types available. Their inclusion here does not constitute the author's endorsement of them. Much of the information about these systems cited here comes from the manufacturers' feature lists. However, the author has offered comments based on his experience (or lack thereof) with given systems. Certainly, the manufacturers' specifications and maintenance literature take precedence over any system data presented here. Given the unusually contentious atmosphere surrounding discussions of network operating systems and particular types of industrial control systems, the author hastens to add that the opinions expressed here are his own, not those of any organization, particularly the ISA, and that he has neither been paid nor will receive compensation of any type for his comments for or against any operating system.

Object-Oriented Programming

Much has been written about what object-oriented programs (OOP) are and are not. Their advantages include reduced development time, better organization of programming efforts, and reusable code. In fact, today any modern network system used in either commercial or industrial applications should be object oriented and have a good "object model" (the structure and interface requirements for classes and objects). OOP programs have classes of objects. A particular manifestation of a class is called an "instance" and to create such an instance is called "to instantiate." The concept of class can be explained by an example; a "class" of objects known as Dog. An instance of the class Dog is an object called Poodle; another instance of this class is called Doberman. They have the same main features found in all objects of the class Dog, but they are different. If you had an instance of the class Dog, you could make your own object by changing the various properties (size, weight, color, hair style, various appendage proportions)—hey, you could have a cocker spaniel. The point is that only the class must be defined, and then all the other objects can be made from it. This object orientedness enables software to be built out of components, rather than be all original or new each time. An automobile is built from components, so also a complex software program. This fact is the basis for the Component Object Model (COM) and now .NET, which are a legacy and current object model respectively.

To be robust, extensible, and scalable, a networked application should have a good object model. In computing, an object is a fragment of code and data that can be summoned or called (like a function) and may be reused, have siblings and children, and generally make life easier for programmers. Objects are also the basis for many other network application software programs such as Sun Microsystems' Java. Objects make possible faster production of programs, more consistency in program operation, less code, and the like. An object can be something as simple as the code for a push button. Using this code you could indicate whether the push button is depressed, has been depressed, or is not depressed. The object code could make the button illuminate, could dictate the alphanumeric on or around the button, locate the button's position on a screen, and determine its size, color, and illumination. All of these parameters are easily amended as attributes of a model and simple data entry (by manual or automated means) can set these values.

Objects can also be as complex as a chart recorder or a PID controller. Microsoft's Visual Basic is an example of a programming language that employs objects. Microsoft has advanced OOP along so that now a program can be built entirely of software components (objects) using a COM server (Microsoft Transaction Server for NT, Component Services for Windows 2000). Those familiar with Windows (2000 or XP) know that most applications are made up of an .exe program (the executable) and usually one or more DLLs (Dynamic Link Libraries). And DLLs are actually objects in the Component Object Model (COM). What Microsoft used to call object linking and embedding has essentially had its name changed to ActiveX. The difference between an .exe and a .dll is that the .exe is referred to as "in process"—same memory space; and the .dll is for "out of process"—running in a different memory space. You may purchase Microsoft's OCX controls (a set of ActiveX components including an executable) or write your own Java ActiveX objects, both of which may very well be a program or a set of programs in an object. Companies use object-oriented programming typically with Sun Microsystems' Java, which assumes that a Java virtual machine is on the device receiving the Java code. JavaScript, which is a scripting language, bears no relation to Java other than similarity in name (now called ECMA Script), and in a C++ programming style. It was developed by an entirely different company and then purchased by Sun Systems. Today, COM and Distributed COM (DCOM) are now "legacy" items. Taking their place, Microsoft now uses .NET, a system based in part on Extensible Markup Language (XML) and what was formerly called Simple Object Access Protocol (SOAP).

An entire set of objects called Object Linking and Embedding (OLE) for Process Control, or OPC, is used in many industrial applications today. We discuss OPC further toward the end of this chapter.

Commercial Systems

Though this is a book about industrial data communications, most of the systems used in the industrial environment had their start in the commercial world. At present, the world is divided into several camps, all of which give the impression that they are "standards" based. Unfortunately, interoperability is a bit difficult.

Stand-alone Systems

Not so long ago, the PC was much like that shown in figure 5-1: not networked and not connected to anything but the occasional bulletin board. Note how self-contained this so-called one-tier PC model was.

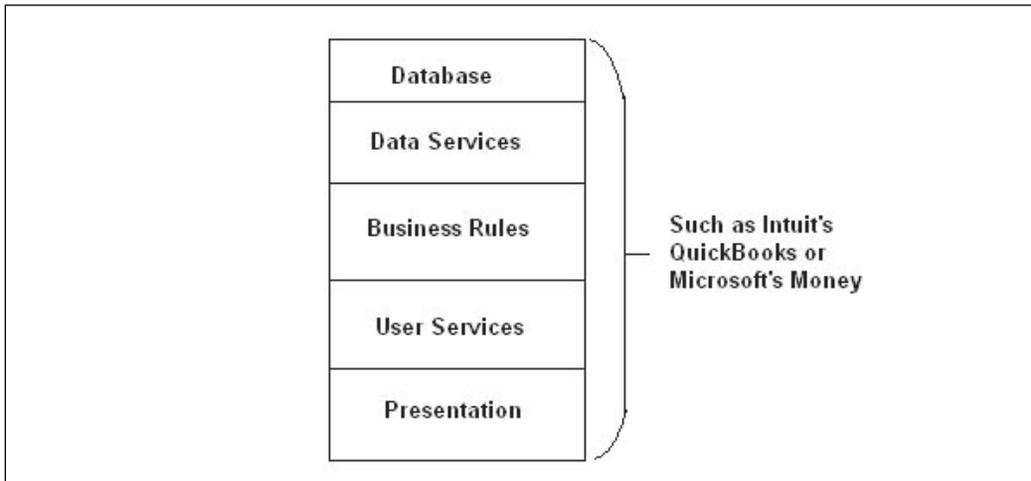


Figure 5-1. Self-Contained (One-Tier) Model

This one-tier model illustrates the way most PC application software was written. A proprietary presentation method (usually a graphical user interface, GUI) would present the information to the user. User services would interface between the business rules and the presentation, usually by presenting hooks (a programmer's way of saying addresses that handle events) to the resident operating system for using the GUI. The business rules would dictate what the user could and could not do with the data. Data services would interface the business rules and the database, which, of course, stores the information. This model would be very much the way a stand-alone analyzer in the industrial area would operate.

Two-tier models, such as client-server, are more complex. Before discussing the client-server model, we need to define these terms. Table 5-1 lists what a server shares. If a device shares a resource, it is a server. A client is a device that uses server resources.

<p>Servers provide the sharing of resources such as:</p> <ul style="list-style-type: none">printersscannersfloppy/hard drivescd-romstape backupsany shared peripheral
<p>Servers provide network resources and network services such as:</p> <ul style="list-style-type: none">communicationse-mailInternetWebprinterdatabasebackupnetwork managementsecurityfile

Table 5-1. Server Resources

Two-Tier Systems

This is the so-called client-server model. Though it may reside on one machine, typically it will be found on two machines, one being a server (any device that shares resources) and a client (any device that uses the shared resources). In figure 5-2, you will notice two diagrams, one for the thick client and one for the thin client. The difference, of course, is where the business rules are located.