[Excerpt from the *NAS Application Style Guide*
Digital Equipment Corporation, 1990]

3

Designing Applications for Integration

Application integration is the ability of two or more applications to work together, in a consistent manner, to perform a task for the user. Like portability, integration is not a black-and-white issue. Rather than speaking of an application being integrated or not being integrated, it is more appropriate to speak of the *degree* of integration with other applications that the application exhibits.

In addition to the degree of integration, you can integrate applications at a number of different points: where the application interacts with the system, with the user, or with other applications.

Section 3.1 explains the principles behind integration and how to maximize the integration of an application. The remainder of the chapter explains the different aspects of integration, the benefits of each, and the services that NAS provides to help integrate your applications.

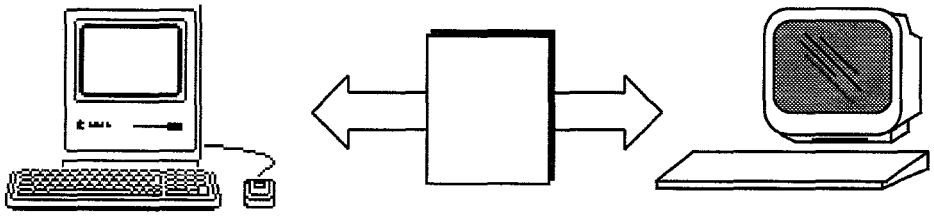
3.1 Understanding How Integration Affects the Application

Because integration is "applications working together," an application cannot be integrated by itself. It requires the cooperation of other applications.

However, integration does not mean that the applications themselves must change or that the applications require an intimate knowledge of each other. For example, you can integrate two existing applications simply by providing a converter that translates the data from one application into a format that is acceptable to the other. This technique is the theory behind the CDA Converter Library. Figure 3-1 shows how you can use the Converter Library to integrate Macintosh paint applications with the DECwrite compound document editor.

Converters are very useful for integrating existing applications. However, converters can affect only the data that the applications produce. To further increase the integration of applications, other techniques are needed.

Figure 3-1 Using Converters to Integrate Applications



3.1.1 Traditional Approaches to Integrating Application Functions

Traditionally, functional integration has been achieved by having one application control the integration. The controlling application contains hardcoded knowledge of the other applications, which it uses to control the operation of those applications. One technique is for the controlling application to use a publically available programming interface to integrate with the other applications. For example, on VMS, applications can use the VAX DEC/Code Management System (CMS) callable interface to integrate CMS libraries into the application. Another technique is to use messaging to send information between applications. (See Section 4.3.3 for an explanation of messaging.)

The problem with traditional techniques for integrating applications is that they require at least one application to have an intimate knowledge of and control over the other application. For example, one application contains calls to the other's programming interface or both applications must understand the same "language" for the messages to be interpreted correctly. By requiring specific knowledge of the other applications, the controlling application must be rewritten each time it wants to integrate with a new application.

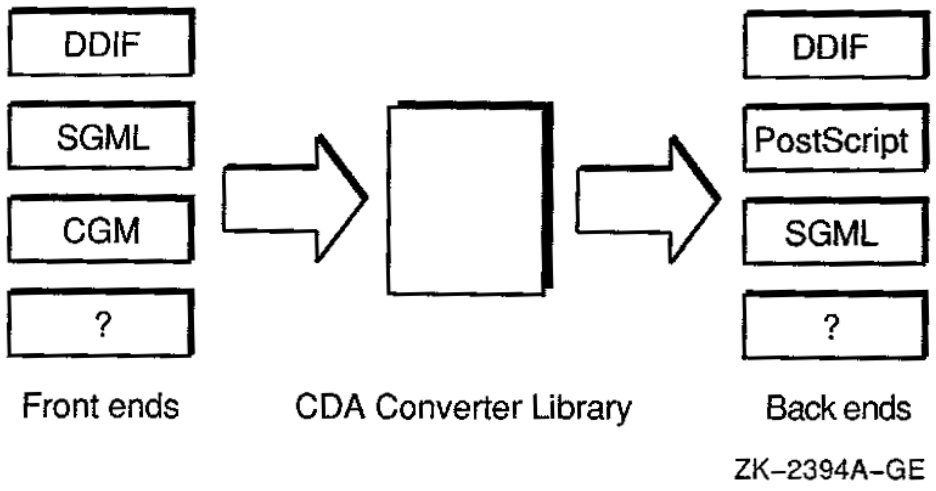
3.1.2 Separating the Communication from the Message

Integration is communication between applications. The reason traditional techniques for integrating applications are so limiting is because the individual applications are controlling the communication, rather than the communication controlling the application. Consequently, to integrate new applications into the system, the controlling application must be changed to incorporate information about the new application.

The solution is to separate the communication mechanism from the message itself. That is, provide a generic mechanism for communicating between all applications. By using a standard communication mechanism, you can add new applications to the system by adding new messages, without having to rewrite the applications themselves.

Separating the communication mechanism from the specific message is the basis of the NAS architecture and is reflected in the architecture for the CDA converter. The converter separates the function (conversion) from the specific request (convert *x* to *y).* The converter then provides a standard interface for both input (referred to as the front end) and output (referred to as the back end). Developers can add new formats - either as front ends or back ends without making any changes to the converter itself. Figure 3-2 shows how the separation of the communication mechanism from the specific message provides powerful extensibility to the CDA converter.

Figure 3-2 CDA Converter Library Architecture



Similarly, for applications to share resources besides data, NAS recommends using a standard mechanism for communicating between applications.

However, to achieve a common communication mechanism, there must be a shift of focus away from the application as an autonomous, controlling unit towards applications as groups of services controlled by the user.