- **Clients**

A major task of client machines is to provide the means for users to interact with remote servers. There are roughly two ways in which this interaction can be supported. First, for each remote service the client machine will have a separate counterpart that can contact the service over the network. A typical example is an agenda running on a user's PDA that needs to synchronize with a remote, possibly shared agenda. In this case, an application-level protocol will handle the synchronization, as shown in Fig. 3-8(a).

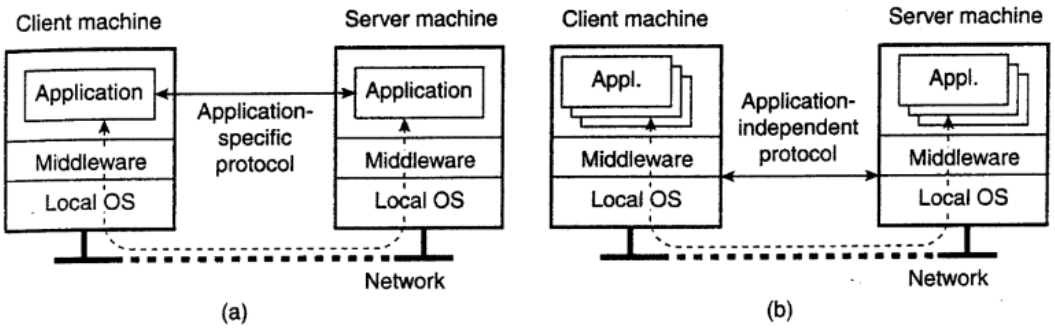

Figure 3-8. (a) A networked application with its own protocol.. (b) A general solution to allow access to remote applications.

A second solution is to provide direct access to remote services by only offering a convenient user interface. Effectively, this means that the client machine is used only as a terminal with no need for local storage, leading to an application neutral solution as shown in Fig. 3-8(b). In the case of networked user interfaces, everything is processed and stored at the server. This thin-client approach is receiving more attention as Internet connectivity increases, and hand-held devices are becoming more sophisticated. As we argued in the previous chapter, thin-client solutions are also popular as they ease the task of system management.

Client-Side Software for Distribution Transparency

Client software comprises more than just user interfaces. In many cases, parts of the processing and data level in a client-server application are executed on the client side as well. A special class is formed by embedded client software, such as for automatic teller machines (ATMs), cash registers, barcode readers, TV set-top boxes, etc. In these cases, the user interface is a relatively small part of the client software, in contrast to the local processing and communication facilities.

Besides the user interface and other application-related software, client software comprises components for achieving distribution transparency. Ideally, a client should not be aware that it is communicating with remote processes. In contrast, distribution is often less transparent to servers for reasons of performance and correctness. For example, in Chap. 6 we will show that replicated servers sometimes need to communicate in order to establish that operations are performed in a specific order at each replica.

Access transparency is generally handled through the generation of a client stub from an interface definition of what the server has to offer. The stub provides the same interface as available at the server, but hides the possible differences in machine architectures, as well as the actual communication.

There are different ways to handle location, migration, and relocation transparency. Using a convenient naming system is crucial, as we shall also see in the next chapter. In many cases, cooperation with client-side software is also important. For example, when a client is already bound to a server, the client can be directly informed when the server changes location. In this case, the client's middleware can hide the server's current geographical location from the user, and also transparently rebind to the server if necessary. At worst, the client's application may notice a temporary loss of performance.

In a similar way, many distributed systems implement replication transparency by means of client-side solutions. For example, imagine a distributed system with replicated servers. Such replication can be achieved by forwarding a request to each replica, as shown in Fig. 3-10. Client-side software can transparently collect all responses and pass a single response to the client application.

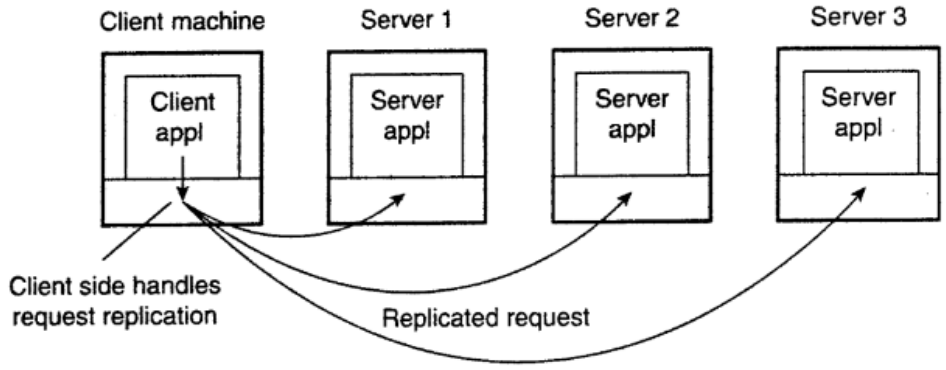

Figure 3-10. Transparent replication of a server using a client-side solution.

Finally, consider failure transparency. Masking communication failures with a server is typically done through client middleware. For example, client middleware can be configured to repeatedly attempt to connect to a server, or perhaps try another server after several attempts. There are even situations in which the client middleware returns data it had cached during a previous session, as is sometimes done by Web browsers that fail to connect to a server.

Concurrency transparency can be handled through special intermediate servers, notably transaction monitors, and requires less support from client software.

Likewise, persistence transparency is often completely handled at the server.