the Availability Digest

www.availabilitydigest.com
@availabilitydig

What's This Stuff Called Middleware?

April 2018

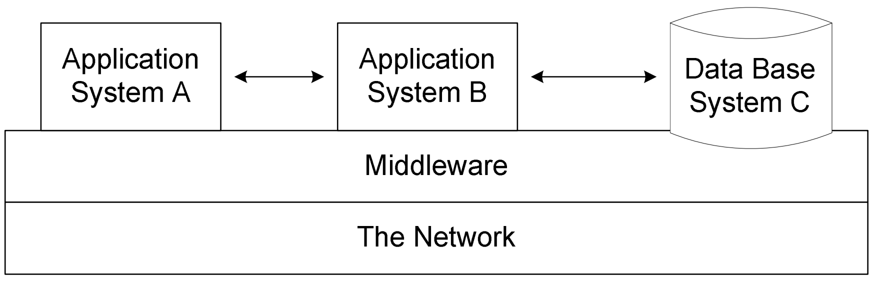
I recently came across an article that I had published in the July/August 1994 Tandem Users' Journal titled "What's This Stuff Called Middleware?" It was published when middleware was just being introduced.

Back then, middleware was seen as the glue that bound client applications to server applications. It was often described as the "/" in "client/server." In my article, I defined middleware as:

> "a kernel running on each computer in the network. The kernels combine to form a logical network and transform the local operating systems into part of an advanced distributed operating system. This infrastructure handles all network I/O for applications, shielding developers from the complex implementation details of supporting multiple operating systems, databases, and transports."

Today's definition is a bit broader. According to Wikipedia, middleware is computer software that provides services to software applications beyond those available from the operating system. Middleware makes it easier for software developers to implement communication and input/output so they can focus on the specific purpose of their application. The term is most commonly used for software that enables communication and management of data in distributed applications.

Middleware is a software layer residing between the applications and databases on various systems and the networking layer which connects these systems into a distributed computing network, as shown in Figure 1.



Middleware's Role in Distributed Computing
Figure 1

Middleware has become so important to today's distributed applications that I thought it useful to review this article some two decades later.

In that article, I defined five types of middleware:

- Transaction
- Message
- Database
- Replication
- RPC

Transaction Middleware

Transaction middleware products provide interactive transaction services between client applications and server applications. If a transaction cannot be delivered to a server, or the server fails to respond, the transaction is deemed to have failed. Transaction middleware is synchronous. The sender generally cannot proceed until the recipient has responded or until the sender declares the recipient as faulty.

Many transaction middleware products provide transaction semantics (Begin, Commit, Abort) so that multiple updates comprising a single unit of work are guaranteed to be made in toto or not at all.

Transaction middleware primarily supports OLTP applications.

Message Middleware

Message middleware provides guaranteed messaging between peer applications. This is generally implemented via store-and-forward mechanisms within the middleware product. Rather than sending a message directly to the recipient as is done by a transaction service, the message makes its way to the recipient via resilient queues. At its convenience, the recipient application can retrieve the message from the network and process it.

Message services are asynchronous. The sender deposits the message in the network and then goes about its business. It does not wait for a response from the recipient.

However, the delivery of the message must be guaranteed, even if the network or server is currently down. Thus, messaging middleware commonly depends on resilient (disk resident) queues.

Messaging middleware is primarily directed at work-flow applications.

Database Middleware

Database middleware allows applications to directly access and update foreign data. This data could be stored in SQL databases or in legacy file systems.

Access to database management systems is typically done by passing command strings from client applications to middleware DBMS servers on the server platform. These are SQL commands for SQL databases. However, more and more database middleware products provide legacy file system access via SQL semantics. These products provide mapping between SQL statements and the corresponding file access commands. A client may open a remote file, position within it, and read, write, update, and delete records.

Database middleware products are most useful for decision support applications.

Replication Middleware

Direct database access and update has the problem that heavy activity from a large number of workstations can present an immense workload to the database host, thus seriously degrading response time. Often, performance can be improved significantly by using data replication. Replication middleware allows data stored in a master database to be copied periodically to user platforms throughout the network.

There is a time period during which the local replicates are out of date (not synchronized) with the master database. This ‘window of synchronization” can range from tenths of seconds to hours depending upon the middleware product, the network capabilities, the number of users, and the application’s needs.

As such, replication middleware is ideally suited for decision support applications.

RPC Middleware

Many of today’s middleware services have come out of extensions to the technology of remote procedure calls (RPC). RPC middleware provides the facilities for a client application to call a remote procedure. These products allow the development programmer to implement a specific set of services that will run on one or more server platforms and then to invoke these services from a client application running on some other platform via an API (application programming interface) also defined by the developer.

These products usually involve a compiler that generates client and server stubs based on the API definition for each desired service. The client stub packages the service request into a message and sends it to the server platform. The server stub converts data formats, passes the request to the service procedure provided by the developer, and returns the result to the client.

RPC services tend to be synchronous and are useful for OLTP and decision support applications.

Many of the other middleware products are RPC-based in that their services are implemented on top of their own proprietary RPC mechanisms. This is particularly true of transaction and database middleware. These products view RPCs as part of their connectivity layer (see Figure 1).

Summary

Middleware technology has now been around for over two decades and has matured significantly since it was first introduced. However, there still rage debates over which flavor is better – messaging middleware or database middleware. Is it better to send a message to a server to access data in a database, or is it better to access that data directly?

Nevertheless, middleware has become a staple of today’s distributed applications. We couldn’t build the complex systems we need without middleware.