PRODUCTS AND SERVICES
- Software Technology Roadmap
- What's New
- Background & Overview
- Technology Descriptions
  - Defining Software Technology
  - Technology Categories
  - Template for Technology Descriptions
- Taxonomies
- Glossary & Indexes
- Feedback & Participation
- Software Engineering Information Repository (SEIR)

Client/Server Software Architectures--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

The term client/server was first used in the 1980s in reference to personal computers (PCs) on a network. The actual client/server model started gaining acceptance in the late 1980s. The client/server software architecture is a versatile, message-based and modular infrastructure that is intended to improve *usability*, *flexibility*, *interoperability*, and *scalability* as compared to centralized, mainframe, time sharing computing.

A client is defined as a requester of services and a server is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration. For details on client/server software architectures see Schussel and Edelstein [Schussel 96, Edelstein 94].

This technology description provides a summary of some common client/server architectures and, for completeness, also summarizes mainframe and file sharing architectures. Detailed descriptions for many of the individual architectures are provided elsewhere in the document.

Technical Detail

Mainframe architecture (not a client/server architecture). With mainframe software architectures all intelligence is within the central host computer. Users interact with the host through a terminal that captures keystrokes and sends that information to the host. Mainframe software architectures are not tied to a hardware platform. User interaction can be done using PCs and UNIX workstations. A limitation of mainframe software architectures is that they do not easily support graphical user interfaces (see Graphical User Interface Builders) or access to multiple databases from geographically dispersed sites. In the last few years, mainframes have found a new use as a server in distributed client/server architectures (see Client/Server Software Architectures) [Edelstein 94].

File sharing architecture (not a client/server architecture). The original PC networks were based on file sharing architectures, where the server downloads files from the shared location to the desktop environment. The requested user job is then run (including logic and data) in the desktop environment. File sharing architectures work if shared usage is low, update contention is low, and the volume of data to be transferred is low. In the 1990s, PC LAN (local area network) computing changed because the capacity of the file sharing was strained as the number of online user grew (it can only satisfy about 12 users simultaneously) and graphical user interfaces (GUIs) became popular (making mainframe and terminal displays appear out of date). PCs are now being used in client/server architectures [Schussel 96, Edelstein 94].

Client/server architecture. As a result of the limitations of file sharing architectures, the client/server architecture emerged. This approach introduced a database server to replace the file server. Using a relational database management system (DBMS), user queries could be answered directly. The client/server architecture reduced network traffic by providing a query response rather than total file transfer. It improves multi-user updating through a GUI front end to a shared database. In client/server architectures, Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server [Schussel 96, Edelstein 94].

The remainder of this write-up provides examples of client/server architectures.

Two tier architectures. With two tier client/server architectures (see Two Tier Software Architectures), the user system interface is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients. Processing management is split between the user system interface environment and the database management server environment. The database management server provides stored procedures and triggers. There are a number of software vendors that provide tools to simplify development of applications for the two tier client/server architecture [Schussel 96, Edelstein 94].

The two tier client/server architecture is a good solution for distributed computing when work groups are defined as a dozen to 100 people interacting on a LAN simultaneously. It does have a number of limitations. When the number of users exceeds 100, performance begins to deteriorate. This limitation is a result of the server maintaining a connection via "keep-alive" messages with each client, even when no work is being done. A second limitation of the two tier architecture is that implementation of processing management services using vendor proprietary database procedures restricts flexibility and choice of DBMS for applications. Finally, current implementations of the two tier architecture provide limited flexibility in moving (repartitioning) program functionality from one server to another without manually regenerating procedural code. [Schussel 96, Edelstein 94].

Three tier architectures. The three tier architecture (see Three Tier Software Architectures) (also referred to as the multi-tier architecture) emerged to overcome the limitations of the two tier architecture. In the three tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. For example, if the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client. In addition the middle layer adds scheduling and prioritization for work in progress. The three tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two tier approach. Flexibility in partitioning can be a simple as "dragging and dropping" application code modules onto different computers in some three tier architectures. A limitation with three tier architectures is that the development environment is reportedly more difficult to use than the visuallyoriented development of two tier applications [Schussel 96, Edelstein 94]. Recently, mainframes have found a new use as servers in three tier architectures (see Mainframe Server Software Architectures).

Three tier architecture with transaction processing monitor technology. The most basic type of three tier architecture has a middle layer consisting of Transaction Processing (TP) monitor technology (see Transaction Processing Monitor Technology). The TP monitor technology is a type of message queuing, transaction scheduling, and prioritization service where the client connects to the TP monitor (middle tier) instead of the database server. The transaction is accepted by the monitor, which queues it and then takes responsibility for managing it to completion, thus freeing up the client. When the capability is provided by third party middleware vendors it is referred to as "TP Heavy" because it can service thousands of users. When it is embedded in the DBMS (and could be considered a two tier architecture), it is referred to as "TP Lite" because experience has shown performance degradation when over 100 clients are connected. TP monitor technology also provides

- the ability to update multiple different DBMSs in a single transaction
- connectivity to a variety of data sources including flat files, non-relational DBMS, and the mainframe
- the ability to attach priorities to transactions
- robust security

Using a three tier client/server architecture with TP monitor technology results in an environment that is considerably more scalable than a two tier architecture with direct client to server connection. For systems with thousands of users, TP monitor technology (not embedded in the DBMS) has been reported as one of the most effective solutions. A limitation to TP monitor technology is that the implementation code is usually written in a lower level language (such as COBOL), and not yet widely available in the popular visual toolsets [Schussel 96].

Three tier with message server. Messaging is another way to implement three tier architectures. Messages are prioritized and processed asynchronously. Messages consist of headers that contain priority information, and the address and identification number. The message server connects to the relational DBMS and other data sources. The difference between TP monitor technology and message server is that the message server architecture focuses on intelligent messages, whereas the TP Monitor environment has the intelligence in the monitor, and treats transactions as dumb data packets. Messaging systems are good solutions for wireless infrastructures [Schussel 96].

Three tier with an application server. The three tier application server architecture allocates the main body of an application to run on a shared host rather than in the user system interface client environment. The application server does not drive the GUIs; rather it shares business logic, computations, and a data retrieval engine. Advantages are that with less software on the client there is less security to worry about, applications are more scalable, and support and installation costs are less on a single server than maintaining each on a desktop client [Schussel 96]. The application server design should be used when security, scalability, and cost are major considerations [Schussel 96].

Three tier with an ORB architecture. Currently industry is working on developing standards to improve interoperability and determine what the common Object Request Broker (ORB) will be. Developing client/server systems using technologies that support distributed objects holds great pomise, as these technologies support interoperability across languages and platforms, as well as enhancing maintainability and adaptability of the system. There are currently two prominent distributed object technologies:

- Common Object Request Broker Architecture (CORBA)
- COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities).

Industry is working on standards to improve interoperability between CORBA and COM/DCOM. The Object Management Group (OMG) has developed a mapping between CORBA and COM/DCOM that is supported by several products [OMG 96].

Distributed/collaborative enterprise architecture. The distributed/collaborative enterprise architecture emerged in 1993 (see Distributed/Collaborative Enterprise Architectures). This software architecture is based on Object Request Broker (ORB) technology, but goes further than the Common Object Request Broker Architecture (CORBA) by using shared, reusable business models (not just objects) on an enterprise-wide scale. The benefit of this architectural approach is that standardized business object models and distributed object computing are combined to give an organization flexibility to improve effectiveness organizationally, operationally, and technologically. An enterprise is defined here as a system comprised of multiple business systems or subsystems. Distributed/collaborative enterprise architectures are limited by a lack of commercially-available object orientation analysis and design method tools that focus on applications [Shelton 93, Adler 95].

Usage Considerations

Client/server architectures are being used throughout industry and the military. They provide a versatile infrastructure that supports insertion of new technology more readily than earlier software designs.

Maturity

Client/server software architectures have been in use since the late 1980s. See individual technology descriptions for more detail.

Costs and Limitations

There a number of tradeoffs that must be made to select the appropriate client/server architecture. These include business strategic planning, and potential growth on the number of users, cost, and the homogeneity of the current and future computational environment.

Dependencies

If a distributed object approach is employed, then the CORBA and/or COM/DCOM technologies should be considered (see Common Object Request Broker Architecture and Component Object Model (COM), DCOM, and Related Capabilities).

Alternatives

Alternatives to client/server architectures would be mainframe or file sharing architectures.

Complementary Technologies

Complementary technologies for client/server architectures are computer-aided software engineering (CASE) tools because they facilitate client/server architectural development, and open systems (see COTS and Open Systems--An Overview) because they facilitate the development of architectures that improve scalability and flexibility.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Client/Server Software Architectures
Application category	Software Architecture Models (AP.2.1.1)
Quality measures category	Usability (QM.2.3) Scalability (QM.4.3) Maintainability (QM.3.1) Interoperability (QM.4.1)

References and Information Sources

[Adler 95]	Adler, R. M. "Distributed Coordination Models for Client/Sever Computing." *Computer 28,* 4 (April 1995): 14-22.
[Dickman 95]	Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.
[Edelstein 94]	Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7,* 5 (May 1994): 34(7).
[Gallaugher 96]	Gallaugher, J. & Ramanathan, S. "Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems." *Information Systems Management Magazine 13,* 2 (Spring 1996): 7-13.
[Louis 95]	*Louis* [online]. Available WWW <URL: http://www.softis.is/> (1995).
[Newell 95]	Newell, D.; Jones, O.; & Machura, M. "Interoperable Object Models for Large Scale Distributed Systems," 30-31. *Proceedings. International Seminar on Client/Server Computing*. La Hulpe, Belgium, October 30-31, 1995. London, England: IEE, 1995.
[OMG 96]	Object Management Group home page [online]. Available WWW <URL: http://www.omg.org/> (1996).
[Schussel 96]	Schussel, George. *Client/Server Past, Present, and Future* [online]. Available WWW <URL: http://www.dciexpo.com/geos/> (1995).
[Shelton 93]	Shelton, Robert E. "The Distributed Enterprise (Shared, Reusable Business Models the Next Step in Distributed Object Computing)." *Distributed Computing Monitor 8,* 10 (October 1993): 1.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Frank Rogers, GTE

Modifications

2 August 97: added reference [OMG 96]
25 June 97: Ed Morris (SEI) updated paragraphs containing information about COM/DCOM
10 Jan 97 (original)
