Client/Server Term Definition

in: Encyclopedia of Computer Science,
D. Hemmendinger, A. Ralston, E. D. Reilly, eds.
International Thomson Computer Publishing, 1998

Silvano Maffeis
Olsen & Associates, Zurich
maffeis@acm.org

December 5. 1997

WHAT IS CLIENT/SERVER?

Client/server is a distributed computing model in which *client* applications request services from *server* processes. Clients and servers typically run on different computers interconnected by a computer network.

A *client* application is a process or program that sends messages to a server via the network. Those messages request the server to perform a specific task, such as looking up a customer record in a database or returning a portion of a file on the server's hard disk. The client manages local resources such as a display, keyboard, local disks and other peripherals.

The *server* process or program listens for client requests that are transmitted via the network. Servers receive those requests and perform actions such as database queries and reading files. Server processes typically run on powerful PCs, workstations or on mainframe computers.

An example of a client/server system is a banking application that allows a clerk to access account information on a central database server. All access is done via a PC client providing a graphical user interface (GUI). An account number can be entered into the GUI along with how much money is to be withdrawn or deposited, respectively. The PC client validates the data input by the clerk, transmits the data to the database server, and displays the results that are returned by the server.

The client/server model is an extension of the *object-based* (or *modular*) programming model, where large pieces of software are structured into smaller components that have well defined interfaces. Components interact by exchanging messages or by *Remote Procedure Calling (RPC)*. The calling component becomes the client and the called component the server. This model creates the advantage of better maintainability and extensibility.

A client/server environment typically hosts various operating system brands and hardware from multiple vendors. Vendor independence and freedom of choice are thus further advantages of

the model. User friendly PC equipment can be interconnected with mainframe based servers, for example.

Client/server systems can be scaled up in size more readily than centralized solutions as server functionality can be distributed across more and more server computers, as the number of clients increases. Server processes can thus run in parallel, each process serving its own set of clients.

The drawbacks of the client/server model are that security is more difficult to ensure in a distributed environment than it is in a centralized one, that the administration of distributed equipment can be much more expensive than the maintenance of a centralized system, that data distributed across servers needs to be maintained consistent, and that the failure of one server can render a large client/server system unavailable. If a server fails, none of its clients can make progress any more.

Furthermore, the computer network can become a performance or reliability bottleneck: if the network fails, all servers become unreachable. If one client produces a high network traffic then all clients will suffer from long response times.

DESIGN CONSIDERATIONS

An important design consideration for large client/server systems is whether a client talks directly to the server, or whether an *intermediary process* is introduced in-between the client and the server. The former is a two-tier architecture, the latter is a three-tier architecture.

The *two-tier architecture* is easier to implement and is typically used in small environments (one or two servers with one or two dozens of clients). However, a two-tier architecture is less scalable than a three-tier architecture.

In the *three-tier architecture*, the intermediary process is used for decoupling clients and servers. The intermediary can *cache* frequently accessed server data to ensure better performance and scalability. Performance can be further increased by having the intermediary process to distribute client requests to several servers such that requests execute in parallel.

The intermediary can also act as a translation service by converting requests and replies to and from a mainframe format, or as a security service that grants server-access only to trusted clients.

Other important design considerations are:

- **Fat vs. thin client:** A client may implement anything from a simple data entry form to a fairly complex business logic. An important design consideration is how to partition application logic into client and server components. This has an impact on the scalability and maintainability of a client/server system.

- **Stateful vs. stateless:** Another design consideration is whether a server should be stateful or stateless. A *stateless* server retains no information about the data clients are using. Client requests are fully self-contained and do not depend on the internal state of the server. The advantage of the stateless model is that it is easier to implement and that the failure of a server or client is easier to handle, as no state information about active clients is maintained. However, applications where clients need to acquire and release locks on the records stored at a database server usually require a *stateful* model, because locking information is maintained by the server for each individual client.

- **Authentication:** For security purposes servers must also address the problem of authentication. In a networked environment, an unauthorized client may attempt to access sensitive data stored on a server. Authentication of clients is handled by using cryptographic techniques such as *public key encryption* or special *authentication servers* such as in the OSF DCE system.

  In public key encryption, the client application "signs" request with its private cryptographic key, and encrypts the data in the request with a secret session key known only to the server and to the client. On receipt of the request, the server validates the signature of the client and decrypts the request only if the client is authorized to access the server.

- **Fault-tolerance:** Applications such as flight-reservation systems and real-time market data feeds must be fault-tolerant. This means that important services remain available in spite of the failure of part of the computers on which the servers are running (high availability), and that no information is lost or corrupted when a failure occurs (consistency). For the sake of high availability, critical servers can be replicated, which means they are provided redundantly on multiple computers. If one of the replica fails then the other replicas still remain accessible by the clients. To ensure a consistent modification of database records stored on multiple servers, a Transaction Processing (TP) monitor can be installed. TP monitors are intermediary processes that specialize in managing client requests across multiple servers. The TP monitor ensures that such requests happen in a "all-or-nothing" fashion and that all servers involved in such requests are left in a consistent state, in spite of failures.

DISTRIBUTED OBJECT COMPUTING

Distributed object computing (DOC) is a generalization of the client/server model. Object-oriented modeling and programming is applied to the development of client/server systems. In DOC, objects are pieces of software that encapsulate an internal state and make it accessible through a well defined *interface*. The interface consists of object operations and attributes that are remote accessible.

Client applications may connect to a remote instance of the interface with the help of a naming service. Finally the clients invoke the operations on the remote object. The remote object thus acts as a server.

This use of objects naturally accommodates heterogeneity and autonomy: heterogeneity since requests sent to server objects depend only on their interfaces and not on their internals, autonomy because object implementations can change transparently, provided they maintain their interfaces.

If complex client/server systems are to be assembled out of objects, then objects must be compatible with each other. Client/server objects have to interact with each other even if they are written in different programming languages and to run on different hardware and operating system platforms.

Standards are required for objects to inter-operate in heterogeneous environments. One of the widely adopted, vendor independent DOC standards is the OMG (Object Management Group) CORBA (Common Object Request Broker Architecture) specification. CORBA consists of the following building blocks:

- **Interface Definition Language:** Object interfaces are described in a language called IDL (Interface Definition Language). IDL is a purely declarative language resembling C++. It provides the notion of interfaces (similar to classes), of interface inheritance, of operations with input and output arguments, and of data types that can be passed along with an operation. IDL serves for declaring remote accessible server objects in a platform and programming language neutral manner, but not for implementing those objects. CORBA objects are implemented in widely used languages such as C++, C, Java, and Smalltalk.

- **Object Request Broker:** The purpose of the ORB (Object Request Broker) is to find the server object for a client request, to prepare the object to receive the request, to transmit the request from the client to the server object, and to return output arguments back to the client application. The ORB mainly provides an object-oriented RPC facility.

- **Basic Object Adapter:** The BOA (Basic Object Adapter) is the primary interface used by a server object to access ORB functionality. The BOA exports operations to create object references, to register and activate server objects, and to authenticate requests. An object reference is a data structure that denotes a server object in a network. A server installs its reference in a name server such that a client application can retrieve the reference and invoke the server. The object reference provides the same interface as the server object it refers to. Details related to the underlying communication infrastructure are hidden from the client.

- **Dynamic Invocation Interface:** The DII is a low-level alternative to the communication stubs that are generated out of an IDL declaration.

- **Internet Inter-ORB Protocol:** The Internet Inter-ORB Protocol (IIOP) allows CORBA ORBs from different vendors to inter-operate via a TCP/IP connection. IIOP is a simplified RPC protocol used to invoke server objects via the Internet in a portable and efficient manner.

- **Interface and Implementation Repository:** The CORBA Interface Repository is a database containing type information (interface names, interface operations, and argument types) for the interfaces available in a CORBA system. This information is used for dynamic invocation via the DII, for revision control, and so forth. The Implementation Repository provides information allowing an ORB to locate and launch server objects.

CLIENT/SERVER TOOLKITS

A wide range of software toolkits for building client/server software is available on the market today. Client/server toolkits are also referred to as *middleware*. CORBA implementations are an example of well-known client/server middleware. Other examples are OSF DCE, DCOM, message oriented middleware, and transaction processing monitors.

- **OSF DCE**: The Open Software Foundation (OSF) Distributed Computing Environment (DCE) is a *de facto* standard for multi-vendor client/server systems. DCE is a collection

of tools and services that help programmers in developing heterogeneous client/server applications. DCE is a large and complex software package, it mainly includes a Remote Procedure Call facility, a naming service, a clock synchronization service, a client/server security infrastructure, and a *threads* package.

- **DCOM**: Distributed Component Object Model (DCOM) is Microsoft's object protocol that enables *ActiveX* components to communicate with each other across a computer network. An ActiveX component is a remote accessible object that has a well-defined interface and is self-contained. ActiveX components can be embedded into Web documents, such that they download to the client automatically to execute in the client's Web browser. DCOM provides a remote instantiation facility allowing clients to create remote server objects. It also provides a security model to let programmers restrict who may create a server object and who may invoke it. Finally, an Interface Definition Language is provided for defining remote accessible object interfaces and composing remote procedure calls.

- **MOM**: Message Oriented Middleware (MOM) allows the components of a client/server system to inter-operate by exchanging general purpose messages. A client application communicates with a server by placing messages into a *message queue*. The client is relieved of the tasks involved in transmitting the messages to the server reliably. After the client has placed a message into a message queue, it continues other work until the MOM informs the client that the server's reply has arrived. This kind of communication is called *asynchronous messaging*, since client and server are decoupled by message queues. MOM functions much like electronic mail, storing and forwarding messages on behalf of client and server applications. Messages may be submitted even when the receiver happens to be temporarily unavailable, and are thus inherently more flexible and fault-tolerant than RPC. Examples of MOM are IBM's MQSeries product and the OMG Event Service. Web *push technologies* such as Marimba's Castanet also fall into the category of message oriented middleware.

- **TP Monitors**: Transaction Processing (TP) monitors allow a client application to perform a series of requests on multiple remote servers by leaving those servers in a consistent state. Such a series of requests is called a *transaction*. The TP monitor ensures that either all requests that are part of a transaction succeed, or that the servers are *rolled back* to the state they had before the unsuccessful transaction was started. A transaction fails when one of the involved computers or applications goes down, or when any of the applications decides to *abort* the transaction. TP monitors are part of client/server products such as Novell's Tuxedo and Transarc's Encina.

  A TP monitor can be used within a banking system when funds are withdrawn from an account on one database server and deposited on an account on another database server. The monitor makes sure that the transaction occurs in a "all or nothing" fashion. If any of the servers fails during the transfer then the transaction is rolled back such that both accounts are in the state they were before transaction was started.

RESOURCES

- *Essential Client/Server Survival Guide*, R. Orfali, D. Harkey, and J. Edwards. John Wiley & Sons, 1994. ISBN 0-471-13119-9.
- *The Essential CORBA*, T.J. Mowbray and R. Zahavi, John Wiley & Sons, 1995. ISBN 0-471-10611-9
- *The TUXEDO System*, Juan M. Andrade et al., Addison Wesley 1996, ISBN 0-201-63493-7
- Client/server news group URL: news:comp.client-server
- Client/server frequently asked questions URLs:
  http://www.abs.net/~lloyd/csfaq.txt
  ftp://ftp.uu.net/usenet/news.answers/client-server-faq.Z
- OMG CORBA documentation URL: http://www.omg.org/
- OSF DCE documentation URL: http://www.rdg.opengroup.org/public/pubs/catalog/dz.htm
- Microsoft ActiveX and related technology URL: http://www.microsoft.com/activex/