Client-Server Design and Implementation Issues in the Accelerator Control System Environment *

S. Sathe, L. Hoff, T. Clifford
Brookhaven National Laboratory
Upton, NY, 11973-5000, USA

Abstract

In distributed system communication software design, the Client-Server model has been widely used. This paper addresses the design and implementation issues of such a model, particularly when used in Accelerator Control Systems. In designing the Client-Server model one needs to decide how the services will be defined for a server, what types of messages the server will respond to, which data formats will be used for the network transactions and how the server will be located by the client. Special consideration needs to be given to error handling both on the server and client side. Since the server is usually located on a machine other than the client, easy and informative server diagnostic capability is required. The higher level abstraction provided by the Client-Server model simplifies the application writing, but fine control over the network parameters is essential to provide the performance required. These design issues and implementation trade-offs are discussed in this paper.

1. Introduction

Very large scale integration and the advent of data communication networks have made desktop computers an affordable alternative to centralized facilities. Data-communication networks connect the computers together, allowing the exchange of information and the sharing of resources between different computers on the network. Resources can now be concentrated in the computer that best provides the resource and that computer can make the resource available to other computers via the network. An application is no longer confined to the resources available on the local computer, but can now use the resources available to the network.

An accelerator control application is an example of such a paradigm. It uses various services such as the database, accelerator device control, alarm handling, data archiving, data display and user interface services. To perform these services one or more of each type of server is available. These servers are distributed across the network and need to be accessible to the application. An application user should be able to access these services on the network without explicitly requesting the network transactions. The computer software should automatically locate the resource and transfer the information to and from the service. In other words, access to the services on the network needs to be transparent. A standard model for such distributed applications is a Client-Server model.

2. Client-Server Model

In a Client-Server model, the server offers the services to the network which the client can access. The term client and server do not necessarily imply computers; they can be thought of as a client process and a server process. In certain cases even a server process may perform a client's role in addition to its server role and vice versa. A Client Server relationship is not symmetrical[1]. This means that they are coded differently. The server is started first and never terminates unless it is forced to.

* Work performed under the auspices of the U.S.Department of Energy

A server typically opens a communication channel and waits for a client request to arrive at the well known address. Upon the arrival of a request, the server executes it in the context of the server process or in a separate one and sends back the results to the client. Then it goes back in the wait state to receive more client requests. The client, knowing the server address, opens a communication channel, connects to it, and then sends request messages to the server and receives the responses. When done, the client closes the communication channel.

A Client-Server model is considered to be part of the session layer and presentation layer of the well-known Open System Interconnect(OSI)[2] Model. This layer hides the application layer from some networking details and differences in data formats between various computer architectures. These higher level abstractions namely Client and Server provide an appropriate interface which makes the distributed application writing simpler.

3. Server Design

A server typically provides a number of services. A service is a piece of code that accomplishes the desired functionality. A Service can be fully defined by its name, input parameters and the results produced. Such a service can be executed in the context of the server process and is called an iterative service, or it can be executed in the context of another process and is called a concurrent service. The iterative services are used when the time to handle a request is known ahead of time. In the case of a concurrent service, the amount of time required to handle the service is unknown or is too long to hold the server process from accepting new requests. Concurrent services need to be reentrant and, if they have to share any global data, a proper locking mechanism is required. Accelerator controls device services such as setting the setpoint of a device or getting the readback from a device are examples of the concurrent type services, as the time required for these services varies with which control device is being used. However, the service that gets server diagnostic information can be of an iterative type of service.

A server can be *stateless* or *stateful*. A stateless server does not maintain any information or state about the clients. However a stateful server accumulates client information to function properly. In the case of a stateless server crash, the client comes to know about it and can retry to contact it. The server can just be restarted and then functions normally. However if a stateful server crashes in the middle of its operation, the server alone has the information to know where to resume operation. Server crash recovery can be complicated. A stateful server also needs to know about a client crash so that it can clean up the client information held with it. An accelerator controls device server that sends back a number of replies for a single client request needs to remember the client address and therefore is an example of a stateful server. However a display server is a stateless server since it does not have to remember any client information.

Another issue in server design is security. Should the server need to identify the client before accepting the request? If the server does employ some identification checking scheme, it should report security faults to some authority. Accelerator control facilities that give control system access to a large user community tend to have some kind of security scheme built in their system.

The issue of heterogeneity is important in the server design. Several kinds of heterogeneity need to be considered: machine architecture independence, operating system independence, software vendor implementation independence and server release independence. Different machine architectures have different data representations. Using higher level languages can solve this problem. The use of standards and portable compilers give the operating system independence. The server release independence implies that the client should be able to run independently of which version of the service is available. Vendor dependencies must be eliminated to increase the portability of the application.

Accelerator controls applications can be written using C or C++ languages to achieve the machine architecture independence. The use of a portable compiler such as GNU (provided by Open Software Foundation) C or C++ compiler gives operating system independence. The use of standard libraries such as POSIX gives vendor independence. In accelerator control applications it is common that a server and/or a client needs to be updated after it has been released. This need may be because of added functionality or a bug fix in the server code. It is often desirable that the old and new versions of server should coexist such that the new server can service the requests from the old or the new clients. The clients should be prepared to use the new server if it exists or should try the old one.

Error reporting is one of the important features of the server. A server needs to return the good or bad status of the service executed. A well defined interface to define all the service-related errors is crucial.

4. Client Design

A client is an entity that requests services from a remote or a local server. The client assembles a request message and transmits it to the server to initiate some action by the server. The first step in a client design is to determine how the client will find the server process to which it wants to send the requests. Some kind of a database is usually employed to hold this information.

The request messages sent by a client to a server can be broadly categorized as send-only, blocked, callback, batch and broadcast[1]. A send-only type message originates at the client end and is sent to the server. There is no reply expected from the server for this message. A client request sent to the display server to update the data to be displayed is an example of a send-only type message. When a blocked message is sent to the server, the client blocks until the reply is received from the server. A request to get the control device server diagnostic information is an example of such a type of message. When a callback message is sent to the server, one or several replies are expected from the server at a later time. To receive such delayed replies, the client now has to become a server and the server has to become a client while originating the replies. For example, an accelerator control device client sends a callback request to a server to receive the data from a device based on a hardware or a software event. A broadcast message is sent to probe the network for servers matching a certain address. The servers matching this address acknowledge the request by sending a reply back to the client. A batch message keeps the requests at the client side until the client lets them go over the network. An advantage of sending requests in batches is that it reduces the network overhead. A single reply for all the requests is sent by the server. Accelerator control clients use batching of request messages to improve overall performance.

There are some issues to consider while determining the timeout values for the client. Servers are likely to take varying amounts of time to service individual requests, depending on factors such as server load, network routing and network congestion. The client should be prepared for the worst conditions or for a variation of service time-outs.

A client can fail to communicate to a server for various reasons. For example, the client may not find the address of the server, or the network between the server and client may not be operational, or the machine on which the server runs may not be up, or the server itself may not be running. The client needs to detect and report these errors in a well-defined fashion.

A client and server running on two computers having different architectures pose a data interpretation problem. To overcome such a problem various strategies can be used. The client can filter the data into a machine-independent format before sending it to the server. The server on receiving the request filters it in its native format. When sending the reply back to the client, the server filters the data in the machine independent format and the client filters it back into the native format.

A second strategy could be that the server always makes the data right after receiving and before sending. This strategy assumes that the server knows about its native architecture data formats as well as the client's architecture data format. Another strategy is that the client always makes the data conversions before sending and after receiving. In this case the client has to know about its native as well as servers's architecture data format. It is also possible to have the receiver always making the data right. In such a case both client and server have to know the architecture of the machine from which the data came. Accelerator control applications can choose from one of the above mentioned techniques that is suitable for their environment. However the technique that converts the data to machine independent format or the case where the receiver always makes the data right are supported by standard industry tools such as RPC[1][3].

5. Client Server Performance

As with any software design, performance is an issue in the design of the server. Numerous client requests can quickly affect a servers's performance, if the server has to do a lot of processing for each request. By keeping the request short and the amount of work required by the server for each request low, the performance can be improved, especially in the case of the iterative server. If the service takes a long time to finish, the server performance can be improved by making it concurrent. If the concurrent service uses a globally shared resource, care should be taken to lock it at the lowest possible level of granularity to avoid delays and assure smooth working of the server.

A client should try to group small requests into one batch and then send it to the server in one network transaction to avoid the overhead involved in sending individual small requests.

One of the parameters that has a big impact on the server performance is flow control. Flow control assures that the client does not overwhelm the server by sending requests at a faster rate than the server can process them. The size of the request message and the rate at which the message is sent need to be tuned for the given network configuration.

Proper network parameter selection is important both on the client and the server side. In the accelerator control applications, the message size typically varies from application to application. It ranges from a few bytes to a few hundred kilobytes. The time required to send and receive the message is mainly dependent on the size of the message for the same distance. It is desirable to be able to set the timeout suitable for a given request. The network receive buffer size for the server is a function of the largest message size, as well as how many clients are expected to communicate to the server simultaneously. The network send buffer size needs to be set as well, depending upon the size of the message and the rate at which they are sent. To help the user to get a handle on the network transaction timing, the client needs to provide the timing statistics for the messages being sent and the reply messages being received.

Last but not least, the network components play an important role in improving the client server performance. High performance network elements such as bridges and routers and high band-width networks, specially for consoles that collect data from a number of front ends, are crucial.

A server health checking mechanism is necessary to be built in the server design. Some diagnostics about the server request handling are highly desirable.

6. Client Server Implementation

One of the major decisions that the implementor needs to make in the beginning is what network transport is appropriate for a given Client Server model. User Datagram Protocol (UDP) and Transmission Controls Protocol (TCP/IP) are widely used transports in accelerator control system. The size of the messages to be exchanged, network topology and reliability of the message delivery are important determining factors amongst many others. UDP seems to be suitable for smaller size messages, typically less than 1000 bytes and for the smaller network. The smaller message size and smaller network ensure a minimal packet loss with normal network traffic. TCP/IP is desirable in case of large message sizes and for the wider networks. It provides a reliable data delivery and also does the flow control so that the sender does not overload the receiver by sending data at a rate faster than it can handle. TCP/IP being a connection oriented protocol, the client needs to reconnect after a server crash.

Having selected the transport, one proceeds to choose the interface to be used to implement the Client Server Model. Remote Procedure Call(RPC) is a well known mechanism that is used to invoke a procedure on a remote system. The RPCs prevent the client and servers from having to worry about details such as sockets, network byte order etc. which makes distributed application writing easier. Some accelerator control system designers choose to write their own RPCs while others utilize the standard ones. Standard RPCs enable the writing of servers and clients in a uniform way. Typically, they provide standard ways for finding the server process on a given host. The standard RPCs provide a mechanism to define the request and reply messages which is vital to any distributed application. Each type of message can be defined by its name. The request and reply data also can be defined in terms of single data items or an arbitrary structure. Errors are handled and reported via a well defined interface. Security mechanisms, both on the server and the client side are provided by the RPC interface. Since RPCs are available on various Unix as well as Real Time systems, the client server code becomes portable. Various machine architecture heterogeneity is taken care of by the standard RPCs. They also provide a uniform health checking mechanism crucial to any distributed application. RPCs provide a mechanism to structure the request and reply data in an arbitrary, user defined fashion. RPCs in general are well suited for synchronous type of communication, where the client blocks until the reply from the server is received. To implement the callback type of message delivery, which is asynchronous in nature, takes extra efforts on the part of the implementor.

Using the concepts described above, a Client-Server model has been designed and implemented for the AGS and RHIC control systems. There are two different implementations, one for each control system, because of different requirements and historic reasons. UDP transport was found suitable for AGS, because of the message size of 512 bytes and a small network of about 40 front ends. As UDP does not support the flow control, the clients needed to introduce the flow control explicitly. As the RHIC supports large message sizes and is planned to have of the order of 150 front ends, TCP/IP was a natural choice. Both the AGS and RHIC accelerator device servers are designed to be stateful. Since TCP is a connection-oriented protocol, the design needed to provide mechanisms for cleaning up the client information from the server as the clients crash. Both iterative and concurrent services are supported by the servers. As the vendor supplied software does not give a handle on the client-server connection timeout, a Unix signal is used to interrupt the system connect call. In the case of a server crash, the TCP-based clients need to reestablish the connection with the server. In contrast, UDP based clients do not have to worry about it. Both blocked and callback type client messages are supported. Client-server implementation is a C++ class library and is portable across Unix and VxWorks operating systems. The class library is based on the standard SUN Open Network Computing(ONC) RPC communication interface. The capability of adjusting the network buffer size and time-outs is also provided. The rpcinfo program supplied by RPC is used for checking the health of the server. To get a handle on more server specific information, the server diagnostics provides information such as start-up time, the machine name on which it is running, the number of synchronous and asynchronous messages it has handled from the start-up time and so on. It also provides the information about callback clients. Typical diagnostic information is as follows:

```
     ADOIF SERVER DIAGNOSTICS INFO
     ----------------------------------------------

Host Name:                acnfec007.rhic.bnl.gov
startupTime:              THU OCT 19 08:30:28 1995

RPC Program Number:          1000002
RPC Version Number:          0
TCP Socket Number:           19
Port Number:                990
Receive Queue Size:          10000 bytes
Send Queue Size:             10000 bytes

Synchronous Messages handled:       2784
Asynchronous Messages sent out:     9178
Asynchronous Active Requests:      328
Async Clients Being Served:        4

Async Client Addresses being used by the ADOIF Server
 ---------------------------------------------------------------

Client Address No 0
----------------------
Host Name:                acnindy04.rhic.bnl.gov
RPC Program Number:    1073742096
RPC Version Number:    1
```

```
Server Port Number:       10998
Process Id:               16272

Client Address No 1
----------------------
Host Name:                acnindy02.rhic.bnl.gov
RPC Program Number:       1073742226
RPC Version Number:       1
Server Port Number:       12741
Process Id:               1402

Client Address No 2
----------------------
Host Name:                acnindy02.rhic.bnl.gov
RPC Program Number:       1073742231
RPC Version Number:       1
Server Port Number:       12744
Process Id:               1407

Client Address No 3
----------------------
Host Name:                acnsun17.pbn.bnl.gov
RPC Program Number:       1073741829
RPC Version Number:       1
Server Port Number:       44807
Process Id:               26005
```

7. Conclusions

The Client-Server Model is a standard model used in the accelerator controls applications. There are various server and client design issues. They include concurrent versus iterative services, stateless versus stateful servers, message security, machine architecture, software vendor and server version independence. The design of built-in mechanisms to send and receive different types of messages such as send-only, blocked, callback, broadcast etc. is necessary. To improve the server performance, proper flow control on the client side is necessary. Selection of network time-outs and selection of proper network buffer sizes is a key to performance tuning. Standard RPCs are well suited to implement a Client-Server model as it addresses most of the design and implementation issues of such a model. A Client-Server model implementation that handles callback type messages tends to be more complex and involved than one that handles only synchronous type messages.

References

[1] J. R. Corbin, The Art of Distributed Applications, Springer-Verlag New York
[2] W. Richard Stevens, Unix Network Programming, Prentice Hall
[3] W. Rosenberry, D. Kenney, G. Fisher, Understanding DCE, O’Reilly Associates, Sebastopol, CA
[4] J. Bloomer, Power Programming with RPC, O’Reilly Associates, Sebastopol, CA