# Microsoft Transaction Server Architecture ®

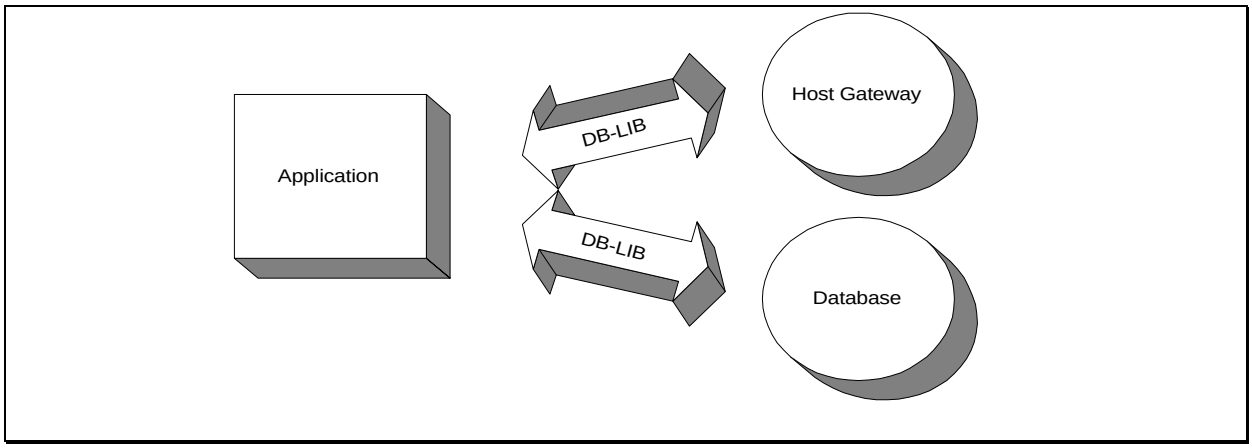
## Transaction Processing

### Background

Traditionally at CTB we have had applications talking directly to the data source in which they were interested. In the case of host applications, we may have used a third tier of a middleware gateway, but the application itself controlled the data flow. This was true for all sorts of applications such as:

- Network Applications. These talked directly to the mainframe and to the database.
- RF Applications. We did not have much success at these applications until we were able to eliminate certain third party middle layers and have direct access to the database.
- Data Transmission Applications. Even these applications, which had no direct contact with the user, were directly connected to the database and the host system.

The following diagram illustrates this connectivity:




### Implementing a Layered Application Model

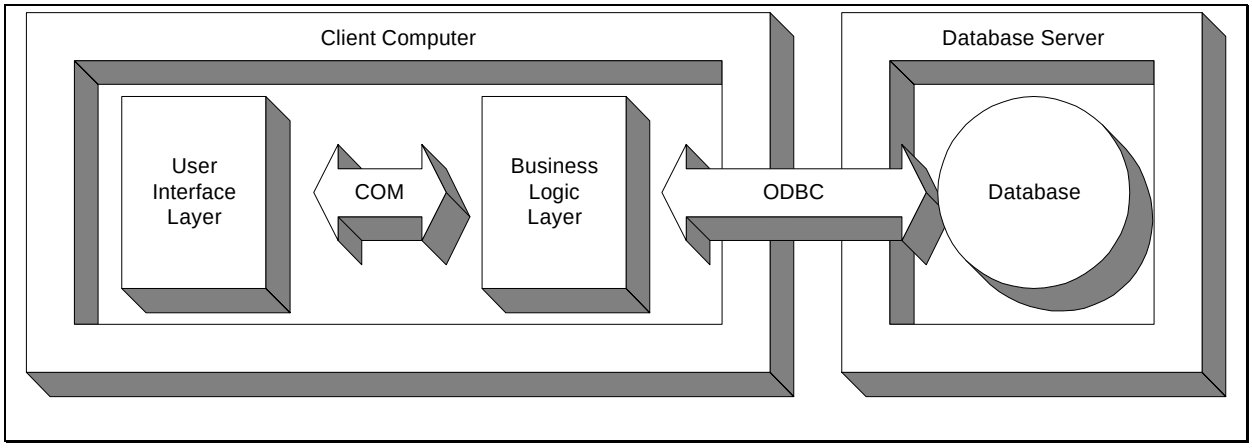
New tools from Microsoft and others have changed the ways that we can create software. Principally, OLE technology has become part of the mainstream. Developers may easily use this technology within their own applications.

There are two parts to this technology that we care about. These are ActiveX components and COM and DCOM communications protocols. ActiveX is the packaging for program objects. When a program is encapsulated in an ActiveX shell, it is accessible to any program that knows how to make calls to an ActiveX® object. The ActiveX object is considered to be embedded in some other application. The diagrams embedded in this document are an example of ActiveX technology at work.

COM and DCOM are communications protocols, much like named pipes or sockets. COM (Common Object Model) is a local protocol for sharing information between two programs in one computer, while DCOM (Distributed COM) is a protocol for sharing information between two programs on two separate computers. Microsoft has made this technology the cornerstone of their interprocess communications

technology. While this is an open standard, with implementations on some non-Microsoft platforms. this is still largely a Microsoft standard. Other vendors in the Windows environment, however, are aggressively implementing these standards. Two of the tools that we are interested in using that support these standards are PowerBuilder and Visio. Many other tools also support these standards. This technology offers us a number of paths to implementing successful systems at CTB clients.

One model is as follows:




While only one connection out of the computer through the Business Logic layer is shown, there may be as many connections as required.

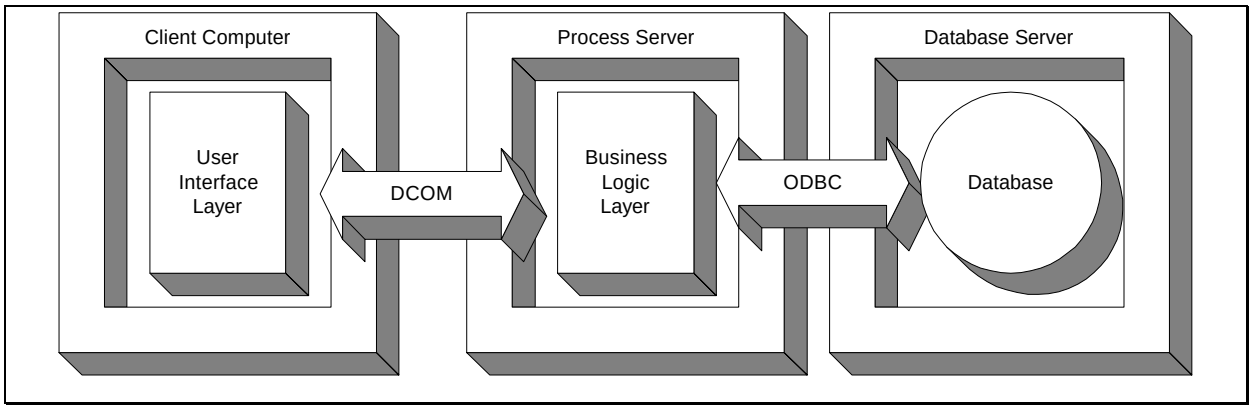
This path is easy to implement and provides us with some advantages:

- The separation between the Business Logic and the User Interface allows us to partition the development task between different developers according to the developers' skill sets.
- We can use best of breed tools for each module, since different tool sets may communicate transparently with COM.
- The Business Logic section needs to be written only once and the code resides in only one place for any function that needs to be called from different User Interfaces (previously known as applications).

## Moving to a Three Tier Environment

Now that we have split apart the different layers of the application, we may move to a more generic model altogether. Rather than forcing all of the application logic to run on one computer, we may partition the application so that application code runs on the most appropriate server.

An illustration of this computing model follows:

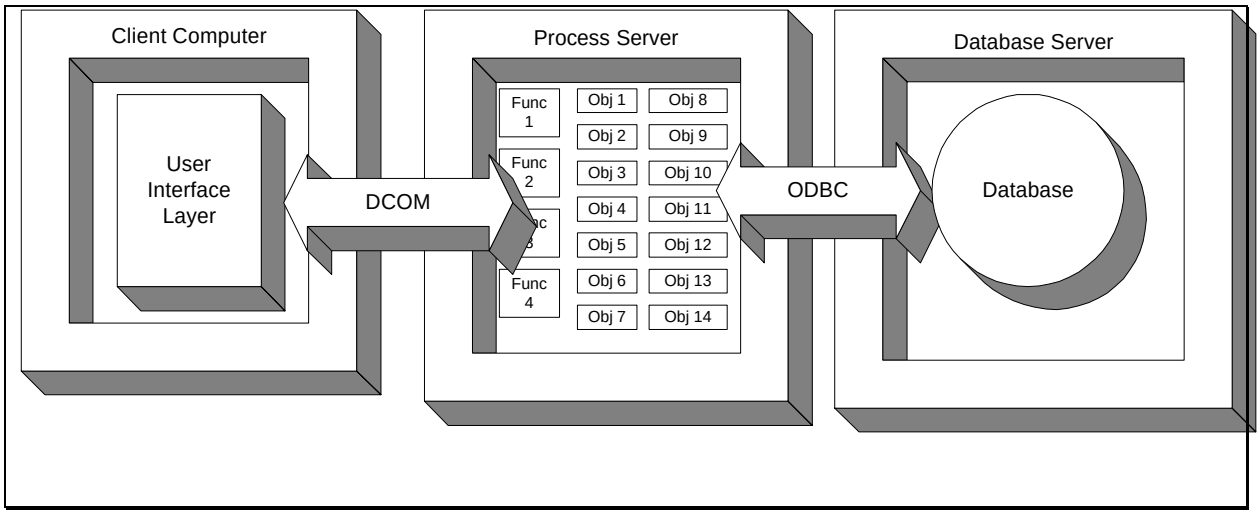



A few of the significant advantages that this model can provide are:

- Use CPU horsepower efficiently. The typical data entry application may have a number of minutes of data entry (which barely taxes the user's computer) followed by a period of intense calculation. During the period of intense calculation, the computer may not be available to the user. By moving the processing portion of the application to a back room computer, one very fast server can be shared by a large number of data entry personnel. This machine performs the data processing, returning the users to productivity quickly. This prevents us from having to continually upgrade the user computers as we add more functionality to the applications.
- Cut down on network traffic. When the application wants to accomplish a complex task, it typically has a dialogue with the servers regarding the implementation of that task. This conversation results in network traffic. If the Business Logic Layer is moved to the same network segment as the database and other servers, less data has to be transmitted outside of the server room. Furthermore, when clients are separated from the network from an RF link, this movement by the application logic onto the network can cut down on radio communications.
- Use common code. Different applications can share common code. All applications that need to perform a particular function can execute the same code. This can ease software maintenance and avoid duplication of effort.

## Expand the Object Orientation

The next logical step in this approach is to move to a more modular-computing environment. This is illustrated here:




The User Interface Layer is provided with a large number of Functions (displayed as (Func *n*), corresponding to distinct business functions. These functions act as traffic directors, calling different application logic objects (Obj *n*) as required.

In previous generations of applications we did some similar partitioning, but we were not able to take it to this level of abstraction. PowerBuilder objects would call application functions that encapsulated the business process for the front-end display logic. The programmer that wrote the front end code did not have to have any awareness of the business performed by the back-end functions. The functions in their turn, called stored procedures to do their work on the database. Stored procedures were a way to encapsulate business data logic in the back-end server, but their language was assuredly not *best of breed*, yielding sub-optimal performance.

A number of developmental advantages come along with this model. Some of these are:

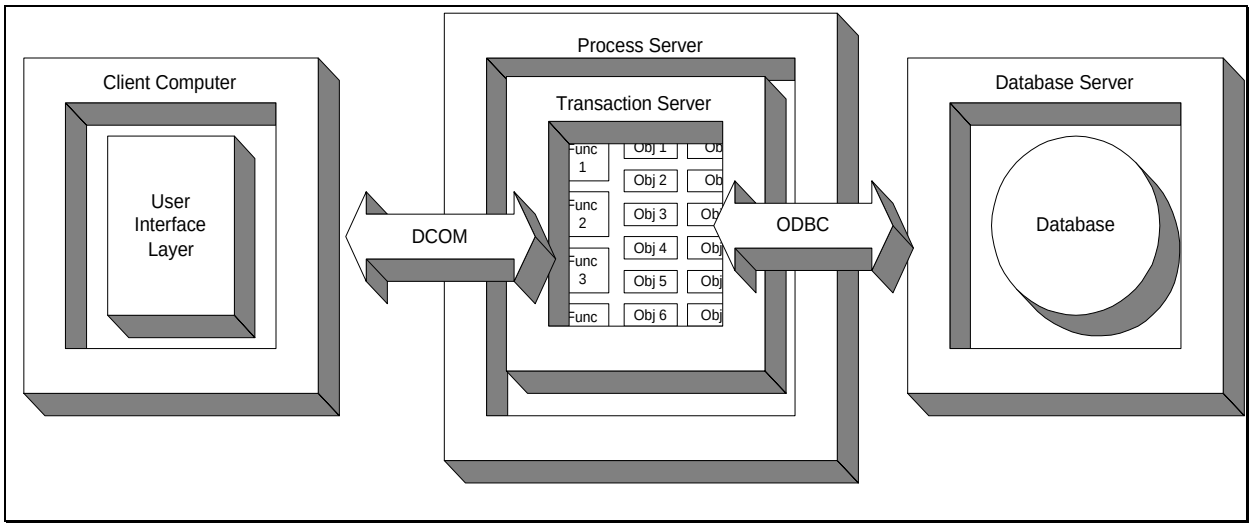
- Object Orientation. This environment lends itself to object oriented code, with its associated reusability and stability advantages. Code is written in small chunks, each of which maps to a particular requirement. These chunks may be tested independently, and then safely implemented in a number of applications with the assurance that the unit function will work properly.
- Ease of Team Development. Different people may safely develop each of the components. As long as the interface between the objects is defined, the developers may control the modules independently.
- Stability with Changes. With each business function encapsulated in its own object, when the business forces an application change, only the affected modules need to be changed. Modules that have not been changed have no requirement to be changed, cutting down on the possibility of inadvertently damaging some other function.

Along with these advantages, moving the functions to another computer brings some significant disadvantages. Each of the functions needs connection to the database and to other resources. The

costs of atomic functions taking and releasing resources may cut performance tremendously. In addition, the computer system that the objects are operating on may be overwhelmed by the calls made to it. It is clear that some sort of management framework is required to meet the needs of this new environment.

Microsoft Transaction Server (MTS) is a management framework that can meet these needs. When we add MTS to our picture, the view looks like this:

## Microsoft Transaction Server




The functionality the MTS brings to the picture is very important. Among its capabilities are:

- Resource Pooling. The MTS provides CPU Threads and Database Connections from pools. When applications ask for a database connection, rather than start a new session with the SQL database, the MTS provides one of the existing connections. This eliminates the cost of establishing the new network database connection. Similarly, processes are given threads from a pool of existing threads.
- Resource Limiting. This same function that provides pooling to enhance performance also provides throttling to keep from bogging down the computer. A limited number of threads or connections may be provided to MTS. When these resources are not available, requesting applications are queued up until they become available. This throttling effect allows MTS to coexist on a computer running other applications, such as SNA Server or even an SQL Database server.
- Transaction Management. Certain transactions may require that different business objects be called within the framework of one business object. The MTS can manage database transactions that span multiple objects. When an object is marked as being transactional, all calls to other objects by that first object are automatically included in the transaction created by the first object. This happens even if some of the objects call different servers.
- Two Phase Commit. This is a fancy phrase that means that all of the servers that are affected by a single business function must either commit or roll back as one group. Even though I have drawn only a single database server, MTS will support multiple servers simultaneously, each with its own pool of connections. Transactions may span multiple servers. If any portions of the

transaction fail, the entire transaction rolls back, reversing any partial processing. Among the servers supported are Oracle, NT SQL Server, MVS CICS, IMS, and more. Any set of servers may be mixed and matched in one transaction.

- Load Balancing. Right now, MTS will allow us to perform load balancing. Just as we used DCOM to move a function to the MTS computer, we can run an object remotely via MTS. This load balancing must be set up by an administrator; as it is not an automatic task. Still, it provides a way to increase the performance of an overall system by adding hardware incrementally as required.
- Security. MTS participates in the NT security, allowing single user log in verification. Access to individual components may be granted based upon the user or the group of which that user is a member.
- Server Management. MTS provides a comprehensive set of tools that shows graphically what is currently happening in the server, which objects are being called, and how many times each one has been called. It has the capability to disable and enable modules as required.

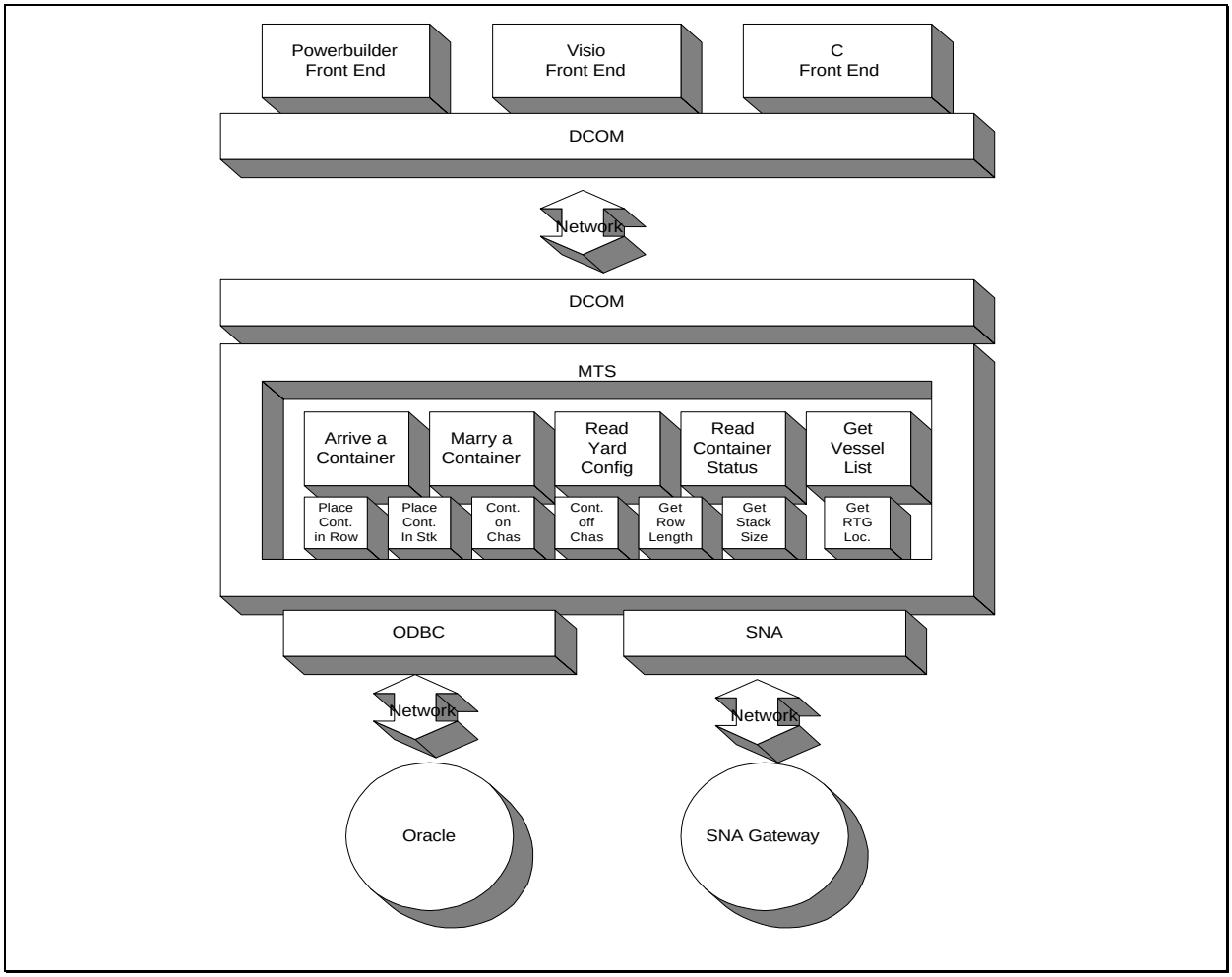
MTS is a developing product. In the future we can expect some significant enhancements. Among the features that we can expect are:

- Automatic Load Balancing. In later releases MTS may be able to monitor the utilization of a number of associated servers and run processes on the most appropriate computer.
- Fault Tolerance. Just as it will be able to monitor the availability of resources on remote computers and respond appropriately, MTS will detect when one of the servers fails completely. It can then stop sending processes to that failed computer, providing transparent fault tolerance to the end user.
- Cluster Participation. MTS may be able to participate in Microsoft's Wolfpack clustering. When one participant in a Wolfpack cluster fails, the users' computers will automatically start using another member as their server. This functionality happens entirely transparently to the end user.

# Transaction Processing at CTB CLIENTS

We see a number of areas that this technology can bring value to the Authority. These areas include network applications, host communications, and replication between the terminals and the central office system. Each of these will be discussed separately below.

## Network Applications




Network applications are the names given to the applications that the users see. It includes all of the desktop functions and Radio Frequency (RF) attached applications that make up the system. The diagram illustrates the architecture used by the network applications.

The principal advantages for us here are in the development, scalability, and network traffic areas.

### Development Advantages

There are a number of development advantages that we have. We can use the most appropriate tool for a particular task. We can use Visio for our yard configurator, while using Visual Basic or


PowerBuilder for the desktop applications. Both of these tools can use the same back end components to query the database about the state of the yard. Visual Basic or PowerBuilder can be used for the back-end component for both modules, where its greater affinity for database operations can be used to best advantage.

One person can concentrate on implementing the visual interface in Visio, which is much more suitable for displaying graphically intense data, like the yard.

PowerBuilder stresses object orientation, and rightly so. This orientation can be built in throughout the application. If we eliminate the capability of the front-end developers from touching the database, the structure can be enforced up and down the line.

## Scalability and Performance

As power is added to the application, more and more processing needs to be done. This leads to requiring larger and larger client workstations. The MTS approach can take us off of this treadmill.

The client application is limited to providing validation and presentation services to the user. All data processing is moved to the back-end process server and database server. Most of the time, the client workstations are sitting idle. We provide the fast processor for those computers to make the short periods of intensive computation as short as possible. Now, we are providing a shared resource of a powerful computer. We are no longer wasting the capabilities of overpowered desktop workstations.

Even this larger powerful computer may run out of steam at some point. We do not need to worry about upgrading many client workstations. Instead, we can add a processor or two to our existing back room computer, add memory to one computer, or even add one more back room computer. This is much more convenient and cost effective than upgrading all of the client computers.

## Network Traffic

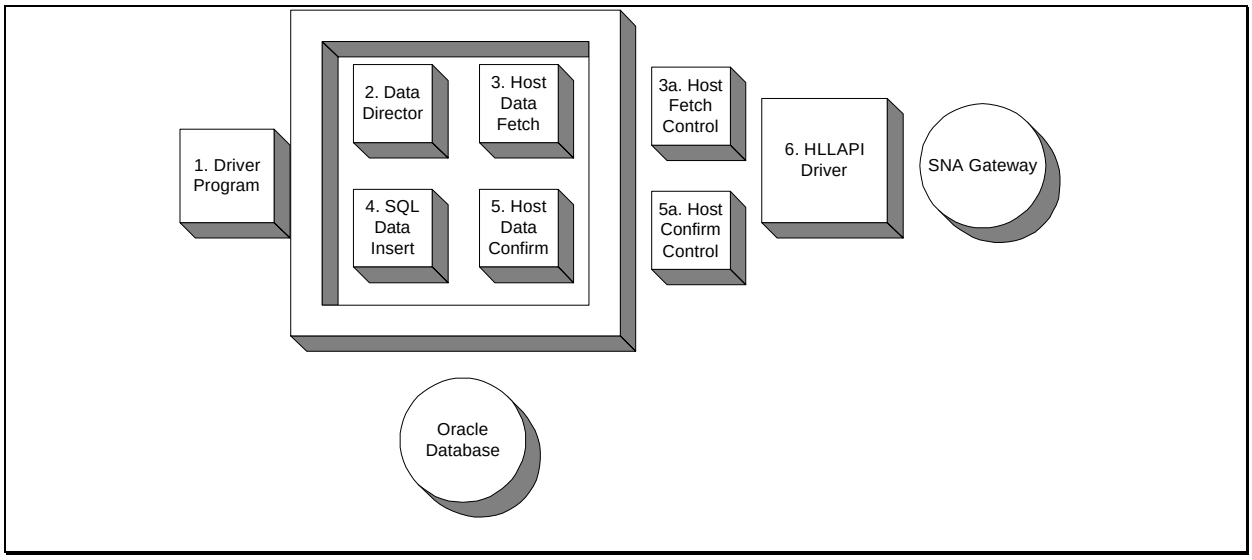
The applications spend a great deal of time passing data back and forth to the database, retrieving intermediate results along with other interactions with the server. This can cause trouble in two situations:

- Radio attached devices. The RF network is slower by an order of magnitude than the wired network. By moving traffic to between the process server and the database server, as well as off of the RF LAN, we can increase performance for an individual user and free up the RF network bandwidth for other users to use.
- Remote Computing during Local Failure. In the event of a failure of a Terminal-based server, we will use a server in the central office for processing. This processing must be accessed using a T1 link, which at 1.54 Mb/sec, is only 15% of the speed of the network. Again, the elimination of the network traffic yields great performance improvement here.

During routine processing, this can make our network much faster. We are going to have a 10 Mb/sec Ethernet network in place to communicate with the client workstations. In the back room, the servers will be communicating using 100 Mb/sec Ethernet. By moving the processing onto a process server in the back room, the middle layer will be able to communicate with the servers at the higher speed.

## Host Communications

We need to perform a great deal of transactional interaction with the host. Information will be flowing bi-directionally. When data is sent, we need to have assurance that the data actually made it from one side of the link to the other.




In both cases, the Client Server end of the connection will initiate the data transfer. The following illustrations indicates the logic of the communications connection for downloads:

The processing involved here is as follows:

1. The Driver Program (1) awakes every set interval of time and requests the Data Director (2) to check and see if any new information is available from the mainframe.
2. The Data Director awakes within the context of the MTS and sends a message to the Host Data Fetch object to see if there is any new information.
3. The Host Fetch Control (3a), which may be on the same computer as MTS, or may be on a different computer, calls the HLLAPI Driver program (6, from a third party) and instructs it to run the program to get new data. If there is no data, the Data Director program is notified and processing ends. Otherwise, the data is returned to the Data Director program.
4. The Data Director passes the data to the appropriate SQL Data Insert control (4), depending upon the type of data passed from the host. The SQL Data Insert control inserts the update into the Oracle Database. If the insert fails, the Director must pass the failure back to the Driver Program for appropriate action to be taken.
5. After the data is successfully written to the Oracle database, the Data Director calls the Host Data Confirm control. Host Data Confirm calls the external Host Confirm Control to mark the transmission as successful on the host.

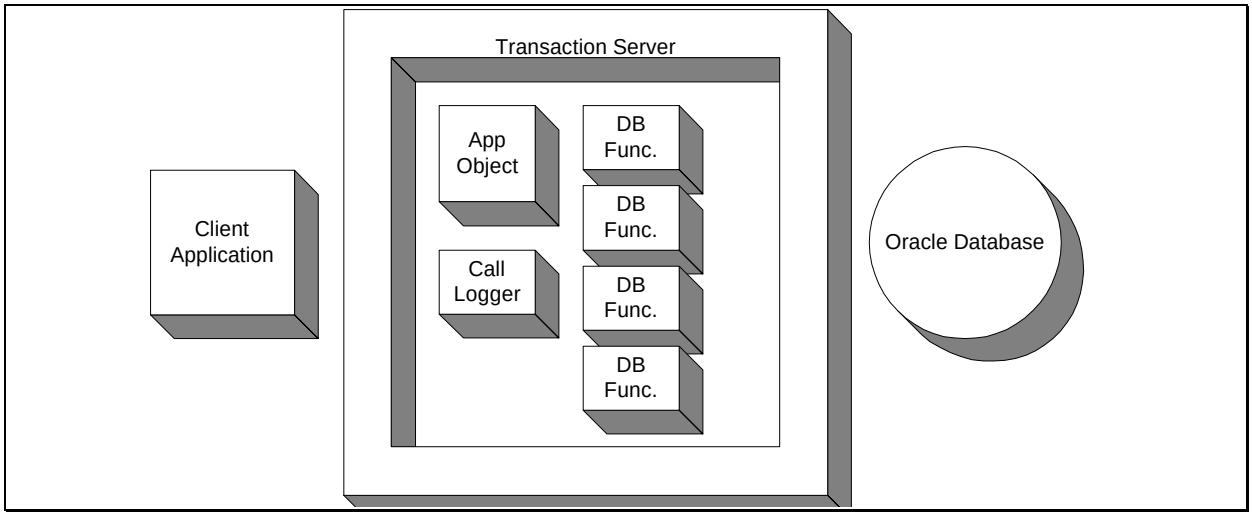
6. After notification that the Host Update has been done successfully, the Data Director notifies MTS that the transaction has been performed successfully. MTS then goes and commits the transaction on the Oracle database.

The important thing to note out of this exercise is that we are enforcing two phase commit logic on a transaction that includes two disparate data sources in an automatic fashion. We are not depending upon application programmers to handle the commit logic.

The flow of data from the client to the host is similar, although in a reverse direction.

## SQL Server Replication

Another area where we see the MTS adding value to our operations is in the area of database replication from one server to another. We have seen the native replication techniques used by the database vendors and we are not impressed. They must be generic and allow for any contingencies. As a result of these constraints, they tend to be overly complex, yet somewhat fragile and hard to maintain. We see that MTS and some custom code can perform the replication for us.

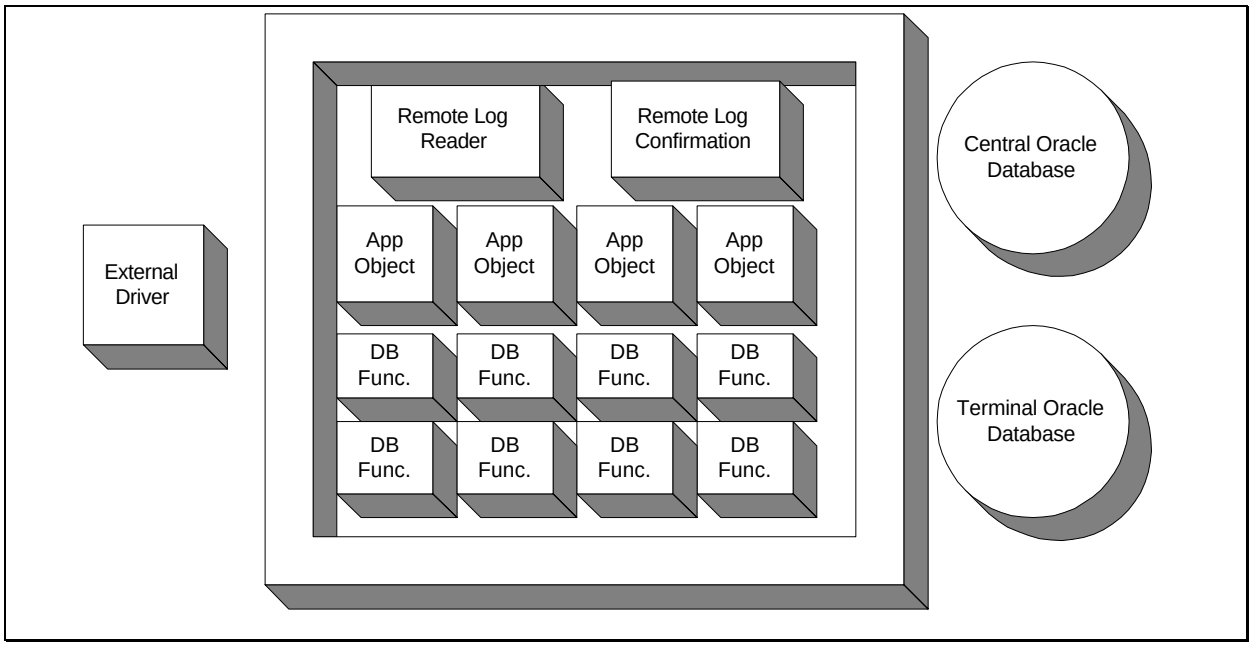



First, we will create an audit trail of all calls to the transaction server. This will be done as part of the transaction process itself, as the following diagram illustrates.

The data flow in every transaction is as follows:

1. The client application calls the appropriate Application Object to perform a function.
2. The Application Object calls the appropriate database functions and/or business functions to perform its required work.
3. The Application Object then calls the Call Logger object to record all of the details of how it was invoked.
4. The entire job is then committed to the database.

At this point, I have a record in a queue (a database table) listing every transaction that is performed against the database.




Next we have a process that moves the data from one database to another. This picture illustrates some of the highlights of this process:

In this scenario, the following steps occur:

1. The External Driver asks the Remote Log Reader object to check the log in the Central Oracle Database to see if anything new has occurred since the last time we looked.
2. The Remote Log Reader finds the piece of work. It loads the name of the called function, along with the associated arguments.
3. Next, the Remote Log Reader calls the *same* function here in the Terminal Oracle Database as was called in the Central database, passing in the appropriate arguments. By just passing the arguments back and forth, the data transmissions are minimized. Additionally, if we wished to perform different processing at the two sites, we would have that capability.
4. After the piece of business is completed, the Remote Log Reader tells the Remote Log Confirmation object that it should mark the line as closed at the Central database.
5. After the remote line is successfully marked as closed, both transactions are committed (two phase commit).

This process can be driven from either end of the T1 line, either pulling the data up from the terminals to the central office, or pushing the data down from the central office to the terminals. If desired, all processing for this function may be centralized with the central office pulling data from the terminals to the central system and also pushing data from the central system to the terminals.


## Conclusion

Microsoft Transaction Manager brings a number of key benefits to the table. Among these benefits are:

- Object Orientation of the Code. Properly implemented, this can substantially decrease development time and radically improve reliability.
- Appropriate Use of Computer Resources. Rather than build up users' workstations so that the rarely occurring busy period can complete a unit of work on the shortest period of time, move the CPU intensive tasks to a powerful processor.
- Best of Breed Tool Approach. Each kind of programming does not require the same development tools. Using ActiveX technology, the best tool for each job may be used.
- Reliability of Data. The ability to arbitrarily assemble a group of objects into a commit group eases the load on the developer to maintain commit logic in their code. Two phase commits guarantee consistency across homogeneous or heterogeneous data sources.

Finally the tools are available for programmers to implement three tier applications without the pain of implementing the low-level communications and control functions themselves.

® Registered Trademarks of Microsoft Corp.