DESIGNER'S OVERVIEW OF TRANSACTION PROCESSING

Lloyd Smith
Tandem Computers
19333 Vallco Parkway
Cupertino, California 95014

Abstract

This paper presents information for the individual responsible for designing a transaction oriented system. It covers the major design considerations that should be taken into account. The paper is divided into the following topics:

I. Introduction

II. A Transaction Processing System Model

III. System Control

IV. Implementation Considerations

V. TANDEM's Transaction Processing System

I. Introduction

One of the important points in any design, and the most important in a transaction environment, is that the system as a whole should be viewed as a "SERVICE". In any service organization the first goal is to find a service that people need. The second goal is to offer a service that people can depend on, and the third goal is to respond to the changing demands of those who use the service. If all three goals are not established from the beginning the success of

NOTE: TANDEM, GUARDIAN, EXPAND, NonStop, and PATHWAY are trademarks of Tandem Computers Incorporated.

the service may be negligible. The one goal most neglected in the past has been responding to change. The combination of user needs, building a reliable system, and the ability to react to change quickly are explored in this paper. The intent of this paper is to offer a better understanding of transaction processing and suggest general guidelines for successful implementations in the future, by examining these three points.

The first design consideration must be to fulfill a user defined need. This need or service is referred to as a "USER FUNCTION". Once a function is offered, a user gains confidence in the service based on reliability and the system's responsiveness to change as the needs of the business change. This ability to change and evolve is referred to as "REACTION TIME". The following design considerations have a direct impact on reaction time.

INSTALLABILITY:

A system can succeed only if it can be installed in a reasonable amount of time. Ease of installation also applies to any major enhancements or user functions.

FLEXIBILITY:

Determines how well the system reacts to change. If a system is designed properly with change in mind, changes can be applied quickly. Reaction time is reduced significantly.

EXPANDABILITY:

Allows the system to accommodate new users and new functions. After a system's initial success, two events typically occur:

1. More users want to use the system. A well-designed system must incorporate the ability to handle an increase in the number of users.
2. New functions are requested. The ability to handle new user functions without affecting the current user functions must also be considered.

MAINTAINABILITY:

The ability to correct problems and "tune" a running application. Too often, this consideration is overlooked in the original design. The problems that occur not only affect the existing functions but future functions as well. If significant resources are required to maintain existing functions, the ability to provide changes and enhancements is reduced. The total reaction time increases and the probability of overall success decreases.

RELIABILITY:

To ensure "service" to the user, the system must be reliable. The best implementations fail if the user cannot access his system. In addition to being constantly available, the system must ensure the integrity of its data base.

Note: Performance considerations are important; however, the above mentioned considerations should not be sacrificed for efficiency. The key to performance is through a good design.

II. A Transaction Processing System Model

Figure 1 shows the components of a typical transaction oriented system. A video display unit provides the human interface to the system. The remainder of the diagram describes the software components of a computer system including a Data Base stored on a direct access device external to the computer. Notice that the flow of control in this diagram is bi-directional. Notice also that operator request may not need the services of all five components. For example, if an operator enters non-numeric data into an entry field defined as numeric data only, the terminal I/O and the field validation routines are the only two components exercised. The request for a new function might involve the display of a new format. In this case four components are exercised: terminal I/O, field validation, data mapping, and transaction control.

This diagram is used as the foundation for the design of a transaction oriented system. The components describe the functional activities that are common to all transaction oriented applications. A brief definition of each component follows the diagram.

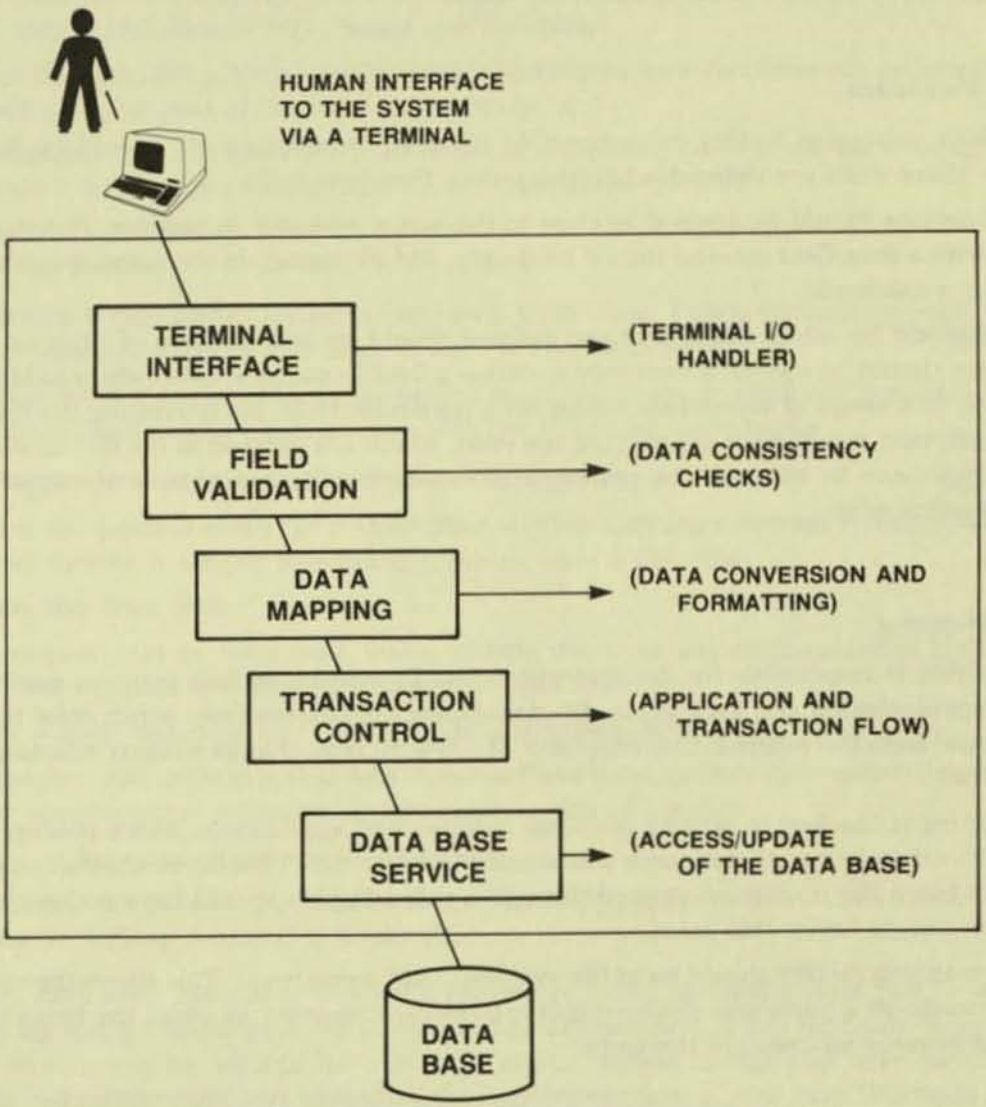

FIGURE 1. TRANSACTION PROCESSING SYSTEM MODEL

Major Program Components

1. Terminal Interface

The terminal interface is responsible for the following general functions:

- All physical terminal I/O
- Control of device-dependent characteristics

Because the physical I/O to various terminal types often involves different protocols, it is advantageous to isolate the code that actually communicates with terminals into one place. This approach enhances the capability to test and install new terminal types, isolate and fix problems, and easily take advantage of new features that may become available on existing terminal types.

The data transmitted by the physical I/O also requires different handling by different terminal types, e.g., the codes to delimit a field may differ.

2. Field Validation

The field validation facility is responsible for data consistency on a field-by-field level. Normally these edits are defined when the screen format is built.

Field validations should be applied as close to the actual operator as possible. Notification of a problem with a data field entered should be timely, and its impact on the running system should be kept to a minimum.

The mechanism by which field edits are defined should be independent of physical terminal type. There should be one consistent way to define a field as numeric data only, a field that must be entered, or a range of acceptable values for a particular field. By separating the type of edit from the physical mechanism of applying the edits, which are defined at the terminal interface, terminal types can be added to the system without altering the logical view of entry fields and their associated edits.

3. Data Mapping

This facility is responsible for the conversion and formatting of data from an external to an internal representation and back again. By developing application tasks which refer to the data in its internal form the external characteristics of a system may change without affecting existing applications.

Data mapping is the key to writing terminal independent applications which process requests with no concern as to how that request was actually constructed. Therefore, whether requests are stored in a batch file on disc or entered through a video display should have no bearing on the application design below this point.

The data mapping facility should be at the symbolic field name level. This allows the ordering of fields to change on a particular screen display; however, the order in which the fields appear in the logical request will remain the same.

4. Transaction Control

This facility is responsible for the overall application flow. It is analogous to the top level implementation of a structured program which:

- Initiates all logical terminal I/O;
- Interprets and validates the request received from the data mapping facility;
- If Data Base access is required to service the request, the appropriate data along with the proper control information is passed to the appropriate Data Base routine; and
- Interprets and validates the Data Base routine replies.

The main function of transaction control is application flow, along with the routing of requests to one or more Data Base routines. It is a relatively small portion of actual application code; however, it is the heart of any application. The major benefits of this approach are:

- Since the actual amount of code necessary to control any one function is relatively small, it is a simple task to add and change user functions;
- Since the approach is structured in a modular fashion, new functions can be integrated and tested easily as part of the whole application; and
- Application flow and control can be tested as a separate piece of the total application and therefore have little or no impact on the Data Base services.

5. Data Base Service

This facility is responsible for all activity on a Data Base. This is normally a very important application function because it alters the state of the Data Base.

A Data Base service routine should be written using the simplest approach possible. The most straightforward and simple approach contains the following components:

- Get a request from a transaction control facility:

  This is the point of entry for a Data Base service. Getting a request from the transaction control facility is similar to reading a record from a disc file,

- Access the Data Base:

  The request may be for a read, write, update, delete or any combination of the four. The specified requests are applied against the Data Base.

- Build a reply based on the results of the Data Base access:

  The reply could contain actual data from the Data Base, control information describing any error condition that occurred, or any combination of the two.

- Reply to the transaction control facility:

  This is the exit point of the Data Base service. Replying to the transaction control facility is similar to writing a record to a disc file.

Moreover, each Data Base service should process requests uniformly from one or more user functions. By doing so the Data Base service will be independent of any particular user function, and the service can be viewed as a general utility, accessible by any user function. This eliminates redundant code and simplifies the implementation of new user functions requiring Data Base services already established.

The Data Base service must be written in a context free environment. The Data Base service should not be responsible for the retention of data between requests. Once a request is received, the Data Base service should be able to process the complete request and then forget about it. This approach simplifies the code significantly and creates an environment that is easy to understand and maintain.

One critical part of any transaction oriented system is input validation. There are three major types of input validation:

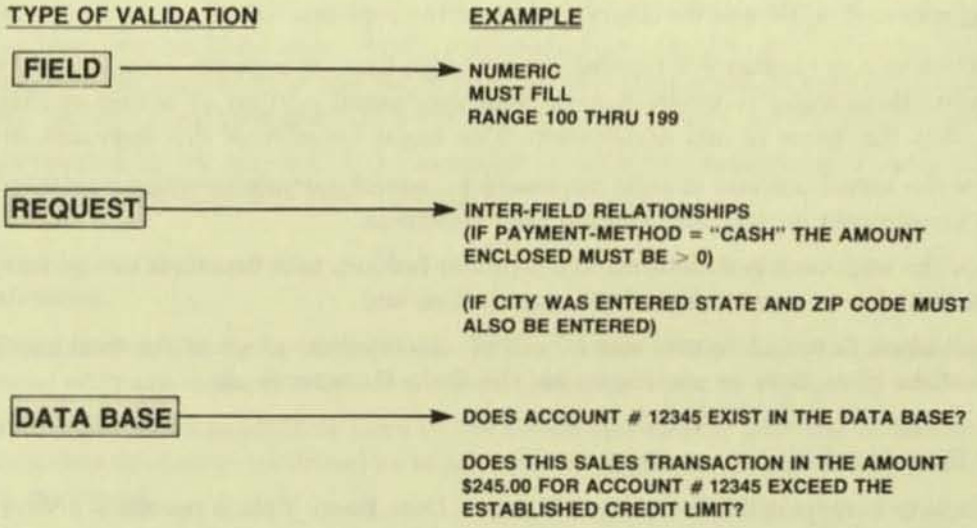

Figure 2 shows each level of edit within the transaction processing system model.

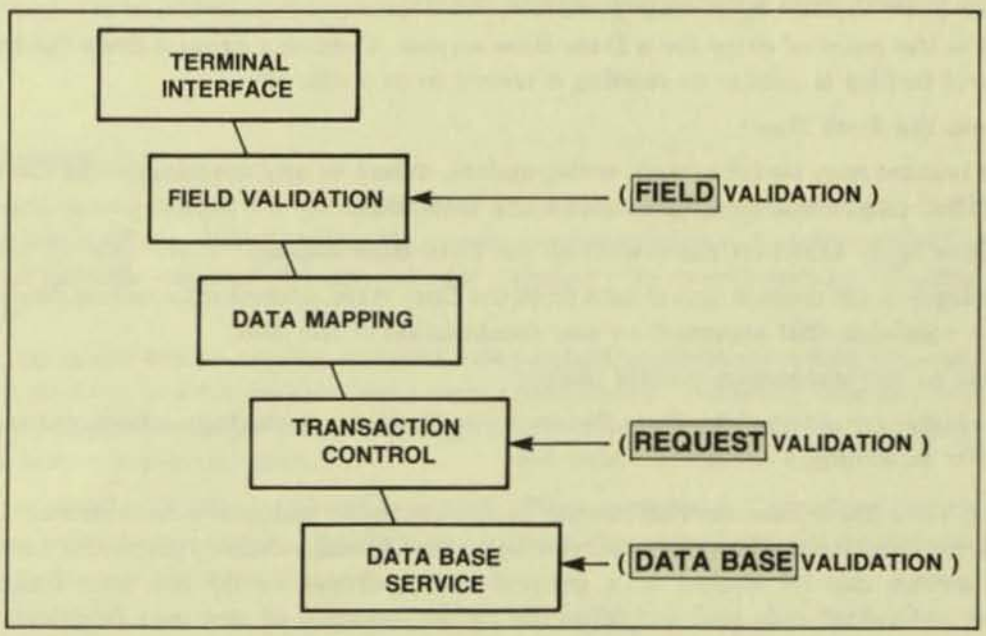

FIGURE 2. EDIT LEVELS

At this point the model is defined. The following illustrations show the use of the model in an application environment.

The internal components of the transaction processing system model can be grouped into the following categories:

Request oriented:
Components 1 through 4 have the combined responsibility for gathering, interpreting and responding to requests. From this point on the combination of components 1 through 4 will be referred to as the "REQUESTOR" portion of our model.

Service oriented:
Component 5, the Data Base services, are written as general utility functions accessible by any user function within a REQUESTOR. The Data Base services will be referred to as "SERVERS".

Figure 3 shows the REQUESTOR/SERVER relationships:

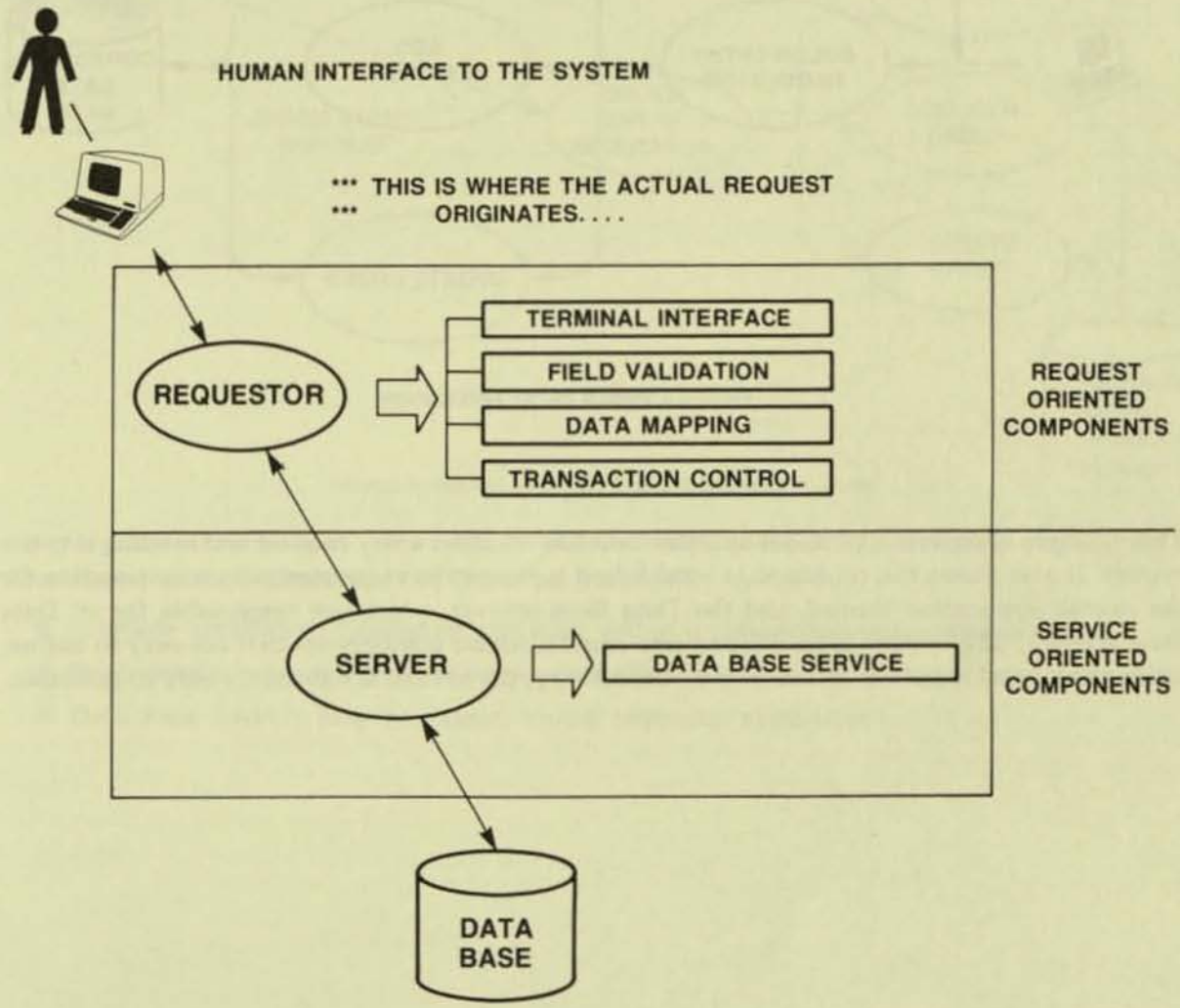

FIGURE 3. REQUESTOR/SERVER RELATIONSHIPS

Figure 4 illustrates an order entry application with three basic functions: credit checking a customer, adding a new order, and updating an existing order.

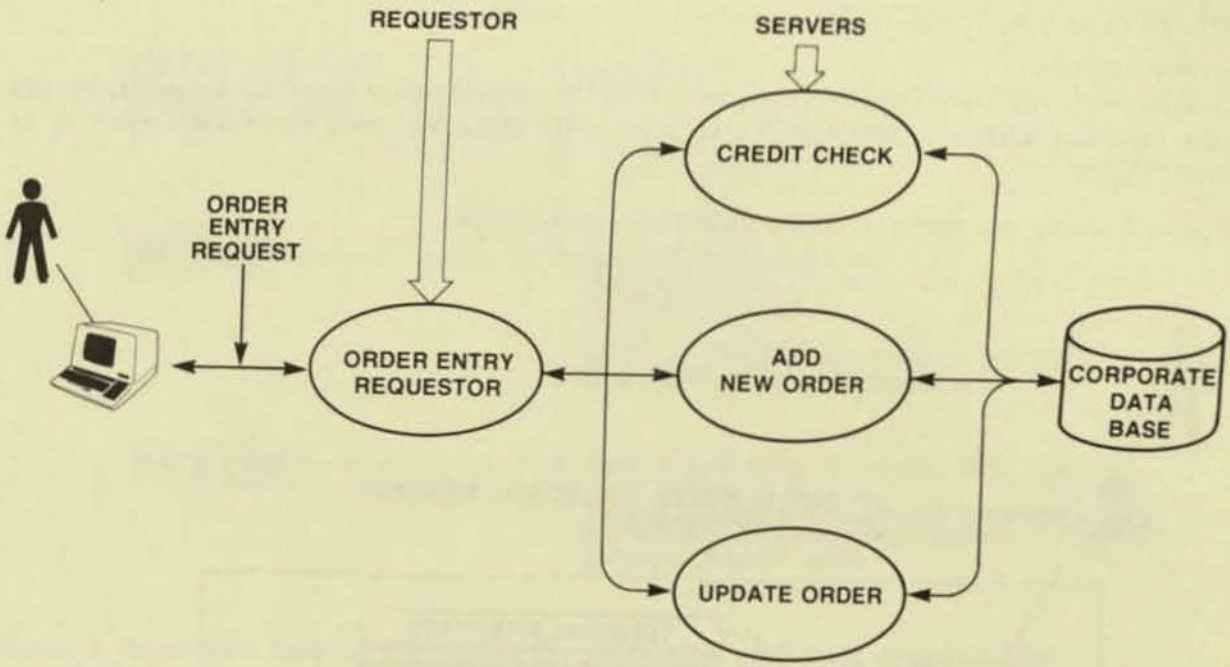

FIGURE 4. ORDER ENTRY APPLICATION

This example illustrates a terminal operator building an order entry request and sending it to the system. It also shows the relationship established between the requestor, who is responsible for the overall application control, and the Data Base servers, which are responsible for all Data Base activity. The system is partitioned into small modular components that are easy to define, write, debug and enhance. Because of its modularity, the system is extremely easy to maintain.

Figure 5 shows the relationships established between multiple applications running under the control of one transaction processing system.

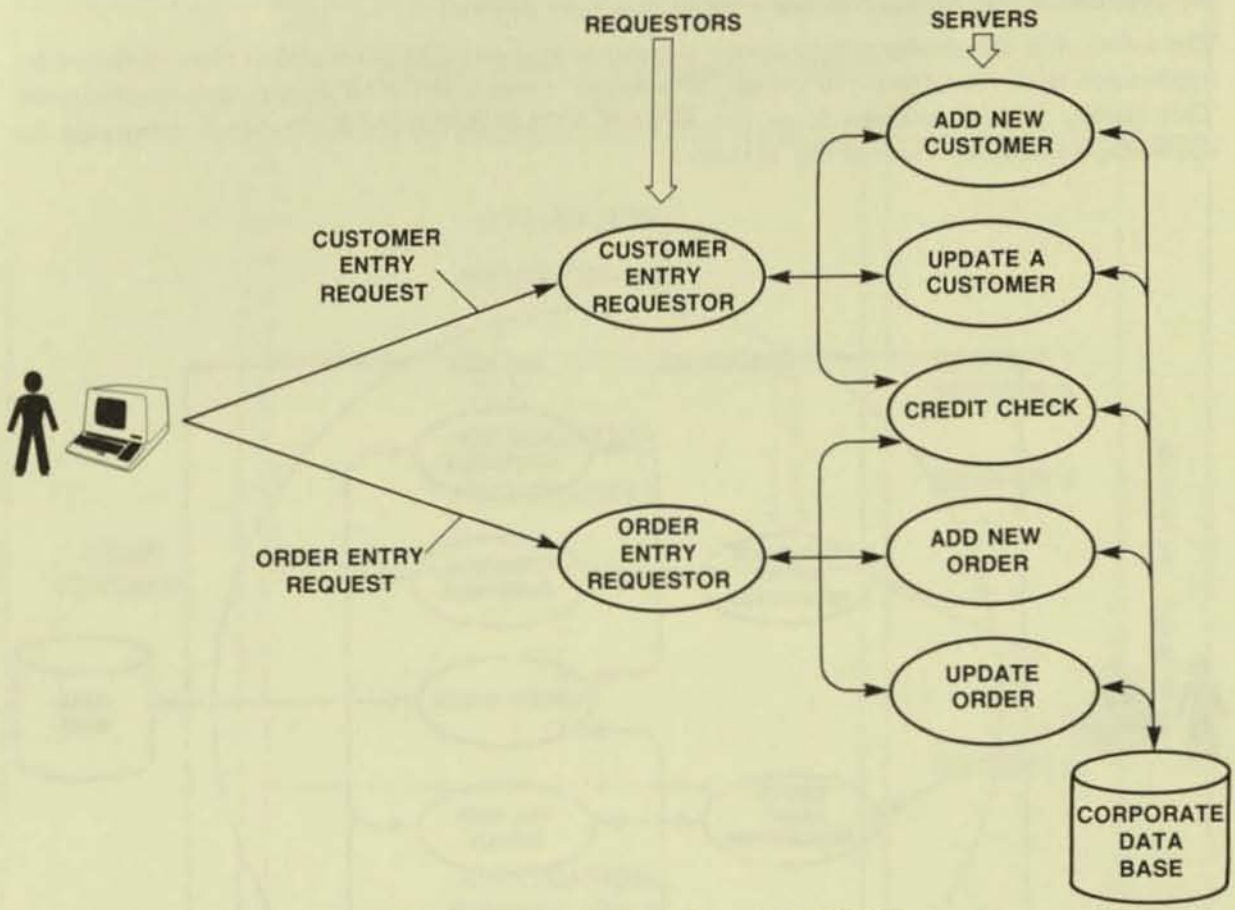

FIGURE 5. MULTIPLE APPLICATIONS IN A SINGLE SYSTEM

The above example illustrates the following points:

- Multiple application requestors can exist within a transaction processing system.
- Each terminal operator should have access to the number of application requestors needed.
- Data Base Servers may be shared among requestor applications.

III. System Control

Taking advantage of modular design techniques improves the overall quality of the system. However, as the components of a system are divided into smaller, more manageable segments, the problem of overall control becomes increasingly difficult.

The solution to this problem is to create a command center that has a global view of the entire application and, therefore, can create, delete and monitor the total application environment. This facility will be referred to as the APPLICATION MONITOR. Figure 6 illustrates the application monitor's view of the system:

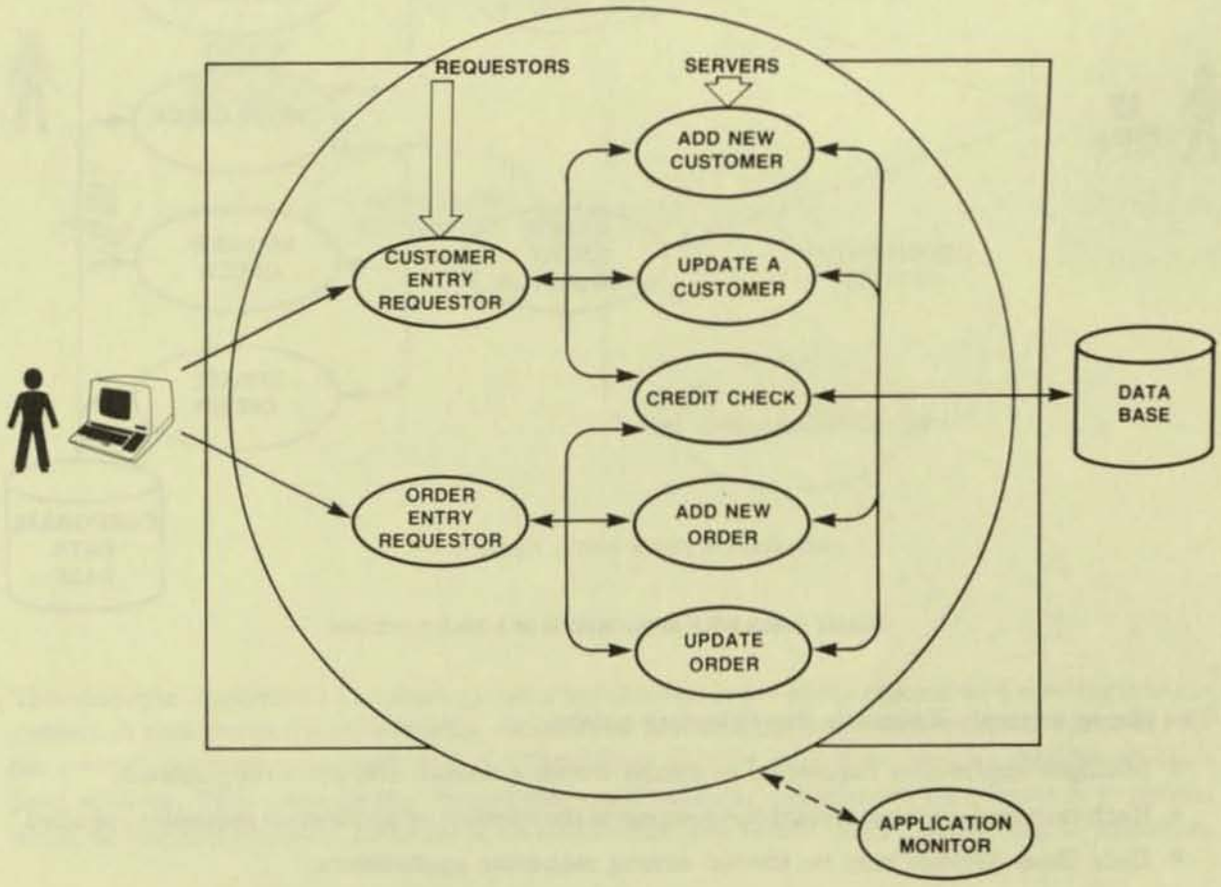

FIGURE 6. SYSTEM CONTROLLED BY APPLICATION MONITOR

The transaction processing system model established previously can be viewed in the following way:

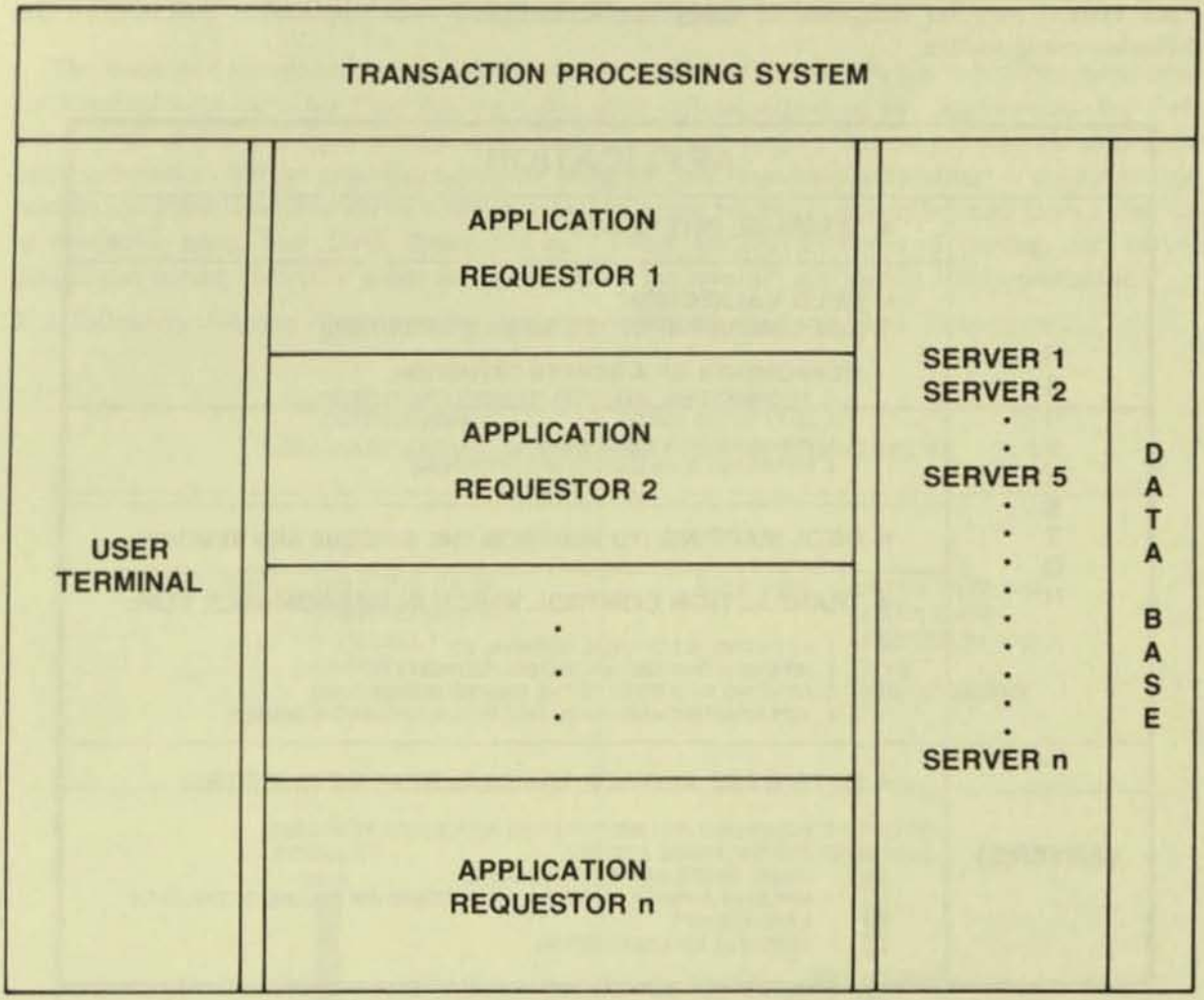

FIGURE 7. TRANSACTION PROCESSING SYSTEM OVERVIEW

The application monitor's view of a transaction processing system is analogous to the view a system operator might have at his console. The system operator controls the hardware environment and the application monitor controls the application environment.

The application monitor should maintain control over the following functions within a transaction processing system:

- Overall Transaction Processing System
- Each Application REQUESTOR
- Each Data Base SERVER
- Each Operator TERMINAL

IV. Implementation Considerations

Once the system is designed, its timely implementation, plus its ability to change as the needs of the business change, will be the deciding factors that determine its ultimate success or failure. Each system may be composed of many applications and each application will require the following components:

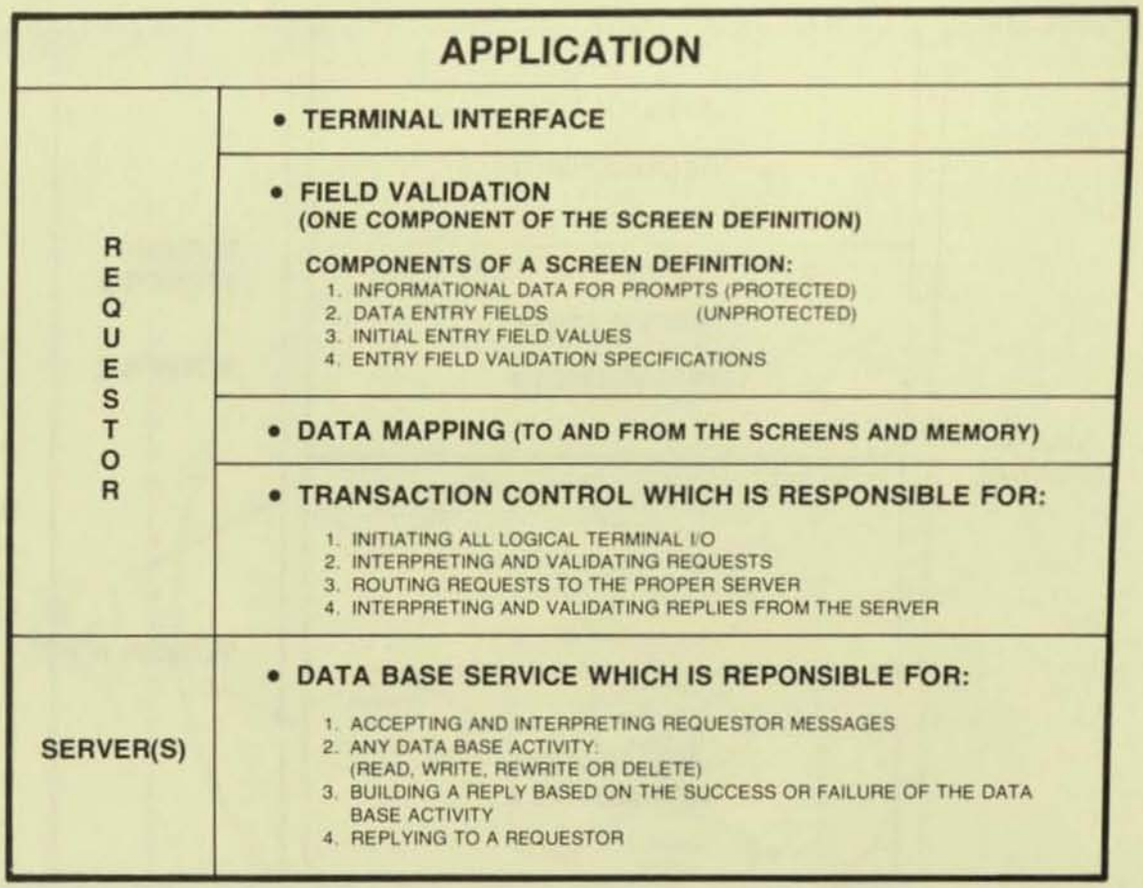

FIGURE 8. FUNCTIONS OF REQUESTORS VERSUS SERVERS WITHIN AN APPLICATION

Requestor Procedures

Because the requestor provides the logic that communicates with the end user, it must be the most flexible component of the system. A means of designing, changing and deleting screen formats is essential. Internal record formats produced through data mapping should be kept and maintained in a data definition library similar to those libraries associated with record definitions on a Data Base Management System. The actual transaction control might be written in a procedural language that is easy to use but flexible enough to handle total application flow.

All the above facilities should be maintained in a library accessible at run time. This allows smooth integration of each function within a requestor. It also allows modular expansion of functions within an application with little or no impact on current running functions.

This approach to implementing an application requestor ensures the reaction time necessary to effectively handle user demand. Moreover, it allows the system to expand in small, well-controlled increments, thus increasing the integrity of the overall system.

Server Procedures

The back-end component of any application is the Data Base server. Back-end functions must be handled with care, for they maintain the most critical aspect of any application, the Data Base. One of the principal advantages of this design concept is that it greatly simplifies the implementation. Server procedures can be designed and tested in the familiar — read a record, update the Data Base and Write a reply — fasion. Input transactions can be read from a disc file or magnetic tape. Test Data Bases can be created for the purposes of testing, and server procedure testing can take place independent of the overall application implementation.

The following diagram illustrates the two step integration of any Data Base server:

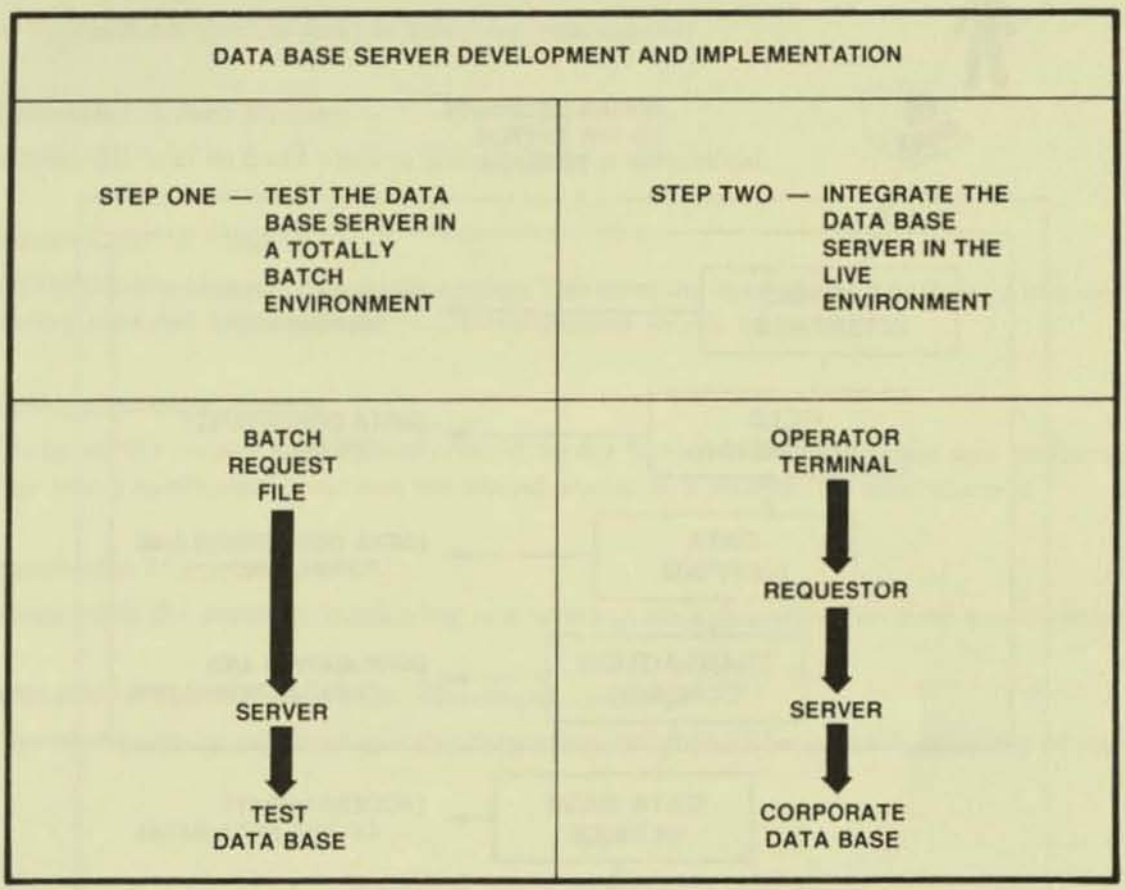

FIGURE 9. INSTALLING A DATA BASE SERVER

The integration of both REQUESTORS and SERVERS into the live system should be handled via the APPLICATION MONITOR. The application monitor should be able to logically start and stop any component within the system.

V. The TANDEM Transaction Processing System

TANDEM offers a total environment for transaction processing. The GUARDIAN OPERATING SYSTEM was specifically designed with NonStop transaction processing in mind. The FILE SYSTEM within Guardian allows separately running processes (in the same CPU, different CPUs within a single system or different systems with an EXPAND network) to communicate with each other at a simple READ/WRITE level. Guardian allows one logical computer system to incorporate up to 16 processors. The EXPAND network allows the interconnection of as many as 255 logical systems within a network and still maintains the simple READ/WRITE level communications between application processes. With this as a base, TANDEM has introduced a new product called PATHWAY. PATHWAY allows a user to take advantage of the unique TANDEM architecture, and it significantly reduces the time necessary to develop a transaction processing system. The functions enclosed within the inner box are addressed by the PATHWAY product.

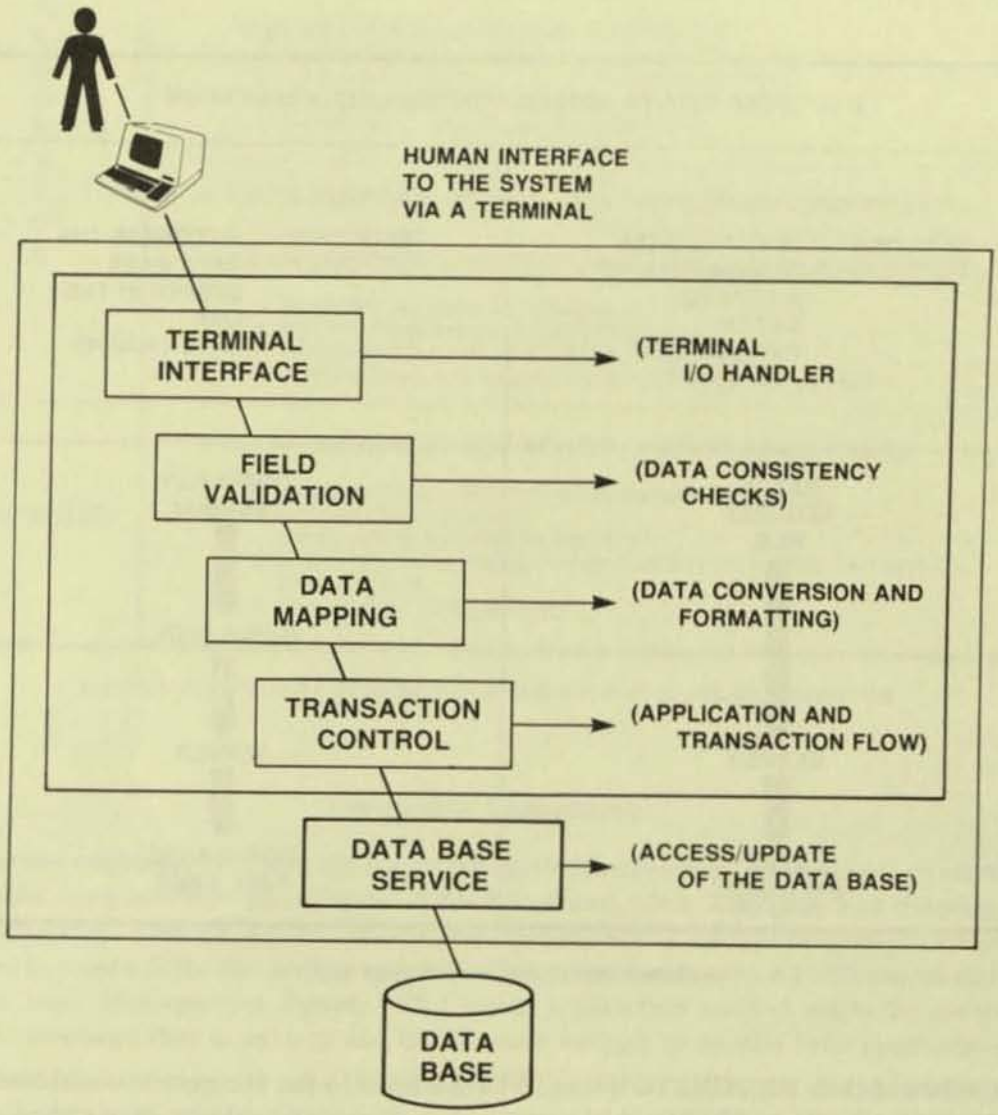

FIGURE 10. THE PATHWAY SYSTEM WITHIN THE TRANSACTION PROCESSING SYSTEM MODEL

PATHWAY Product Overview

The goal of the PATHWAY product is to simplify the design and development of transaction oriented applications. The PATHWAY product addresses four of the major components necessary to implement a transaction oriented application.

1. Terminal Interface	(Multi-terminal I/O handler)
2. Field Validation	(Data consistency checks)
3. Data Mapping	(Data conversion & formatting)
4. Transaction Control	(Application & transaction flow)

The fifth component (Data Base Server) can be implemented using any of the TANDEM standard languages — COBOL, FORTRAN, TAL, or MUMPS.

The PATHWAY product has the following components:

- *Interactive Screen Builder*

  Allows the user to build screens interactively at a terminal.

- *Screen COBOL compiler*

  A COBOL-like terminal oriented language. The compiler creates and maintains a pseudo code library accessed by the Terminal Control Process at run time.

- *Terminal Control Process*

  Interprets the pseudo code-library created by the Screen COBOL compiler and performs the four major application functions mentioned above in a NONSTOP environment.

- *Application Monitor*

  Responsible for creating, monitoring and altering the application run time environment.

- *AMCOM – Application Monitor Command Language*

  The mechanism by which an operator may communicate with an active Application Monitor.

If we assume a successful installation of a transaction oriented system, we now must deal with EXPANDABILITY. By using the unique TANDEM architecture, an application can be written and then expand smoothly as the demands placed on the system increase. Most successful systems first expand because of an increase in the number of users who need to use it. Figure 11 shows the addition of new users and interjects a new question: "Do I run more than one copy of

the total application or do the users share the application?" Figure 11 shows users sharing the application. It should be noted that TANDEM's terminal control process, a part of PATHWAY, handles all the multi-tasking between more than one terminal of the same type. In Figure 11, notice that the application remains unchanged even though the number of users increases.

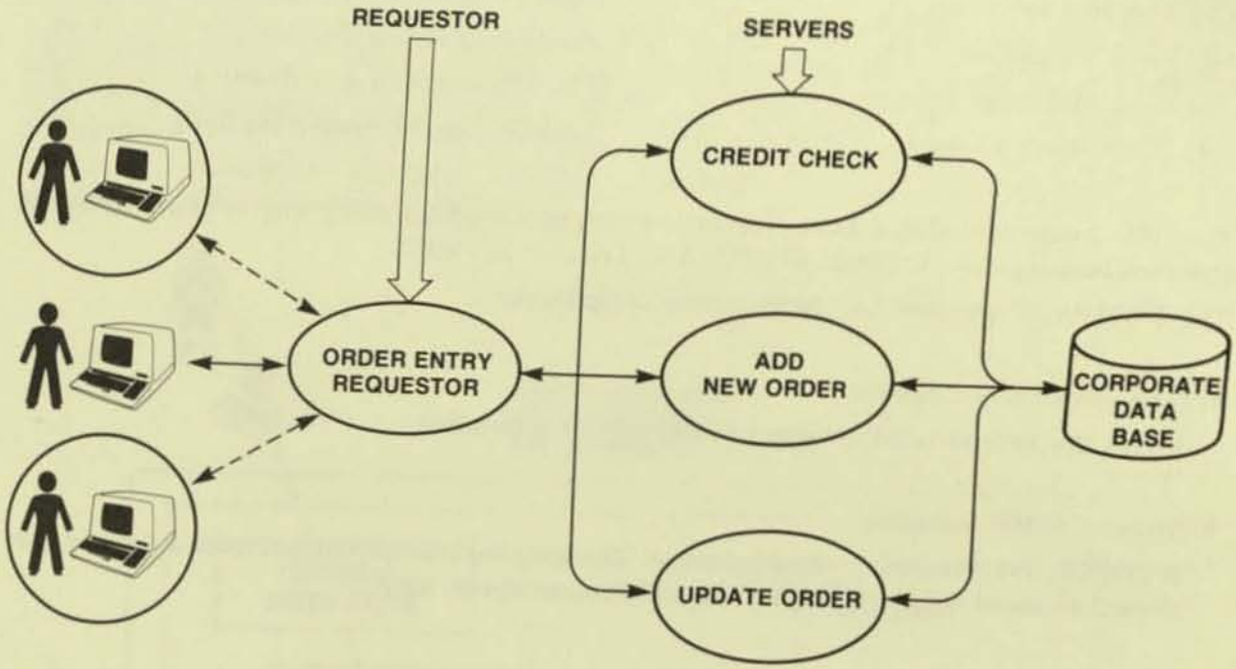

FIGURE 11. ADDING USERS TO PATHWAY SYSTEM

In a successful application, the addition of users can degrade system performance. At that point, most systems go through either a major change or a rewrite. The unique TANDEM approach offers an alternative.

To alleviate the problem of increased user load, the system must be able to distribute the application into more than one physical process. This is the key to expandability.

The description of the system to this point views the system as one self-contained application. However, all expansion limitations can be eliminated if one logical application can comprise multiple physical processes or running programs. This leaves the original design of the system unchanged, but increases system throughput by expanding the modularity of the application beyond the physical boundaries of a single program unit.

The following diagrams illustrate the various ways of functionally distributing the application to allow for expansion:

- Terminal operators can be connected to multiple requestors running the same application.
- Multiple copies of the same server can be created to increase throughput.
- Sharing a server between more than one application requestor.

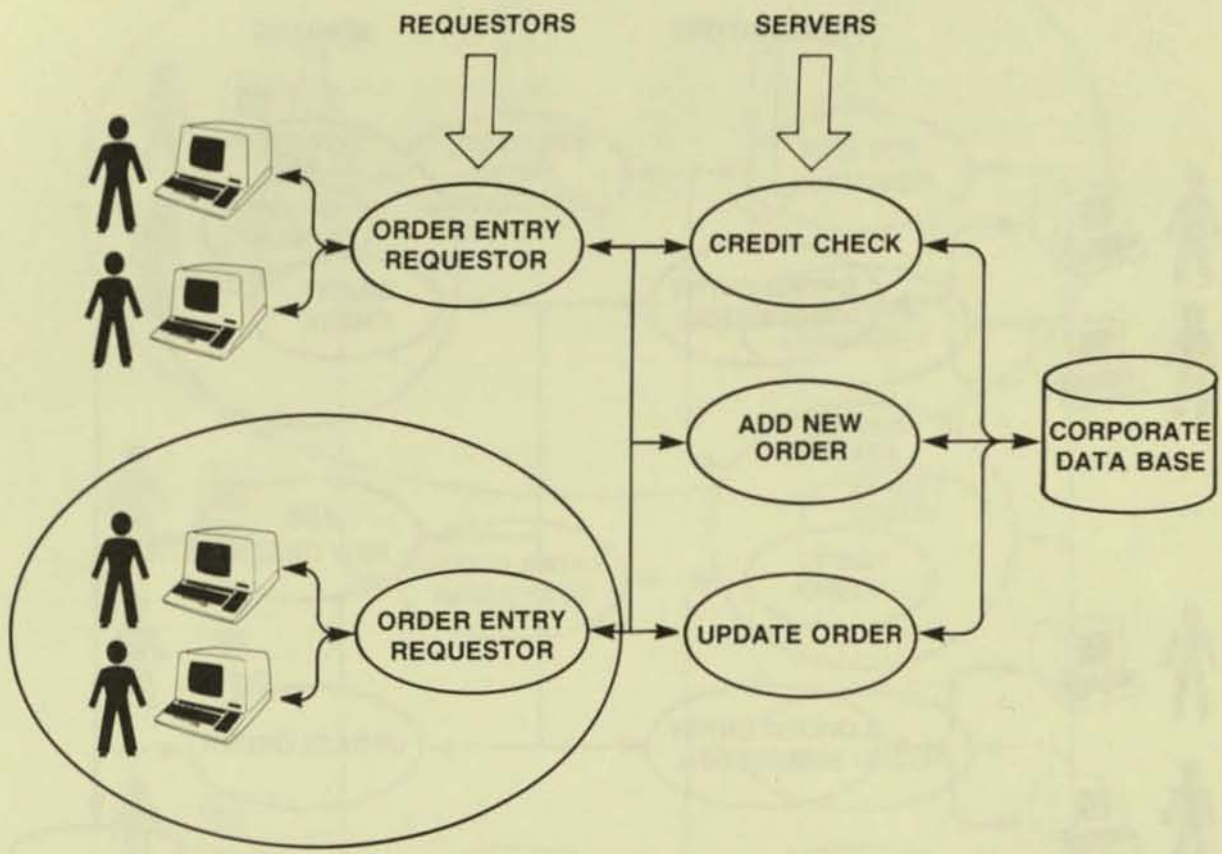

FIGURE 12. SYSTEM EXPANSION BY ADDING REQUESTOR

Notice that the users are distributed between two requestors and that the requestors share the server functions. Therefore, the total number of servers remains constant even though the system includes multiple requestors.

Another expansion problem is caused by increased demand on any one server. This problem can be overcome by duplicating a particular server to create what is known as a "SERVER CLASS". For example, if the majority of user demand on the system is to run credit checks, using a single server for credit checking could create a bottleneck and impact the total applicaton. This bottleneck can be eliminated by creating another copy of the credit check server and distributing the requests between the two copies. Figure 13 illustrates a server class:

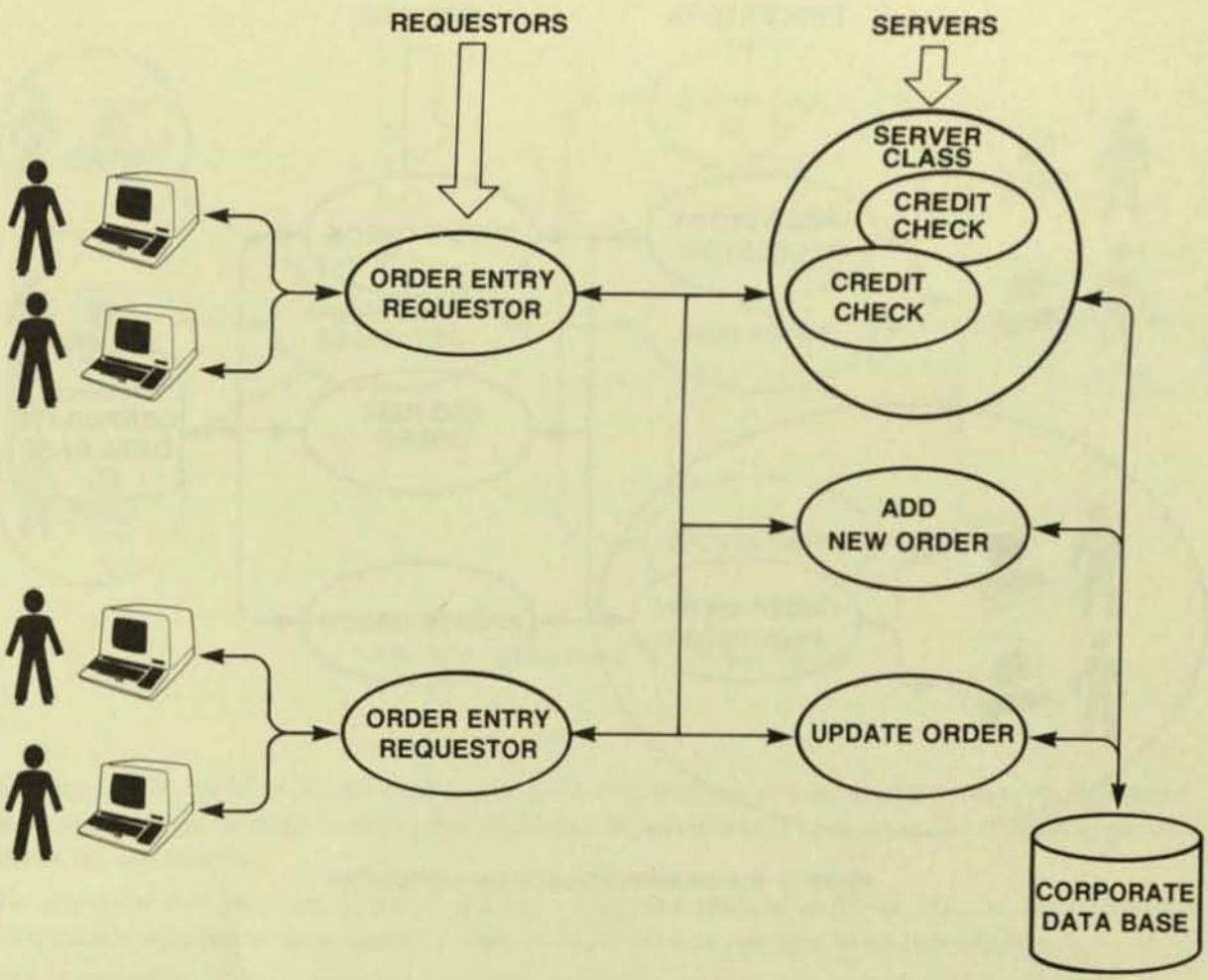

FIGURE 13. SERVER CLASSES

Figure 14 illustrates the addition of a new application. Notice that even though there are two unique applications, the requestors can still share servers or server classes.

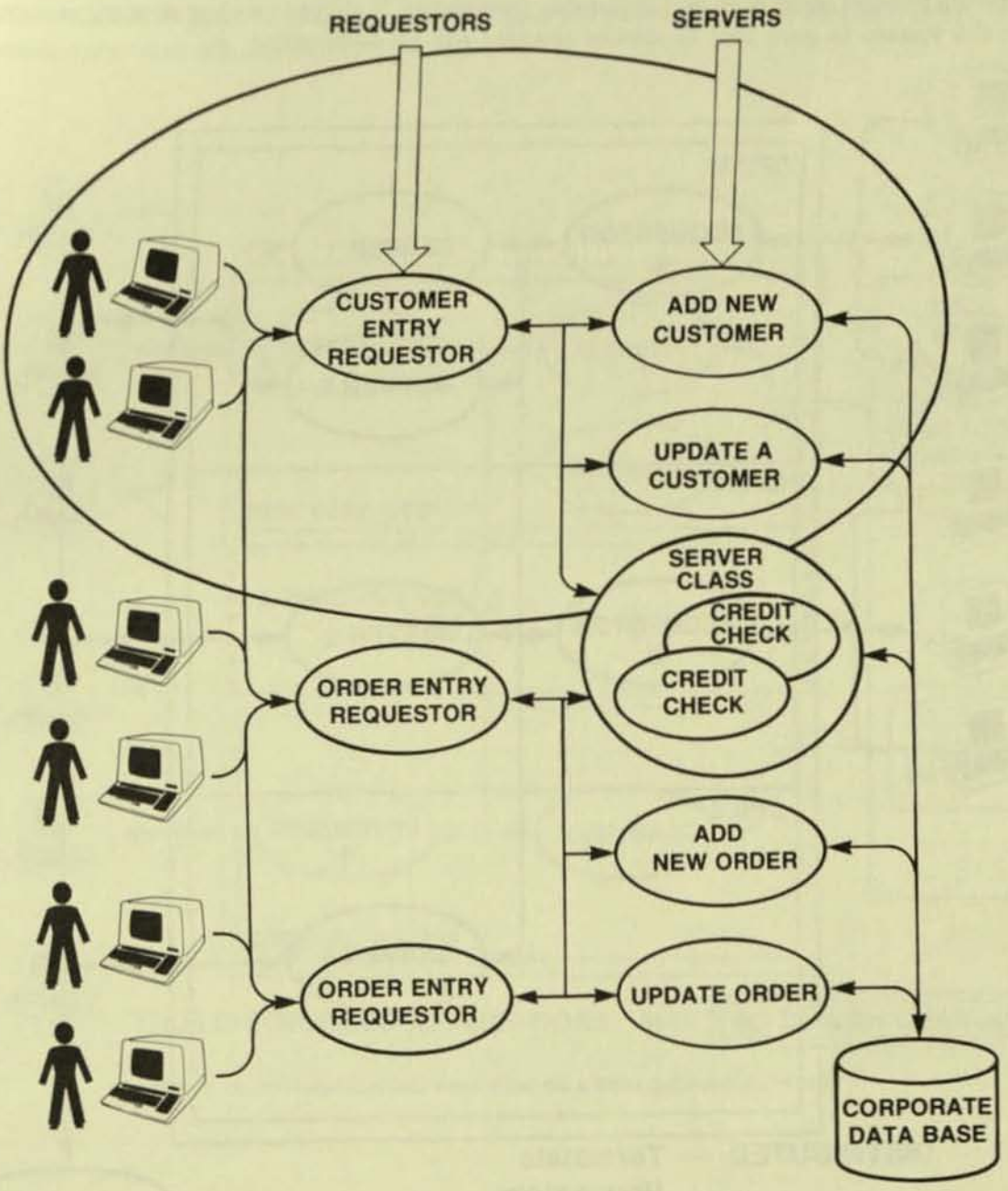

FIGURE 14. ADDING A NEW APPLICATION REQUESTOR

Figure 14 shows that both the order entry and customer entry applications need to be able to check credit. The credit check server class is therefore shared by both applications, yet each application also has its own private servers to fulfill the individual requirements of a particular application.

One of the major reasons for distributing an application into multiple processes was for expandability. Figure 15 illustrates the distribution of application functions within a local TANDEM system. Terminals, Requestors and Servers may be distributed among multiple CPU's, maximizing parallel operations and increasing throughput. It should be clear that when demand forces the system to grow that no design changes will be necessary.

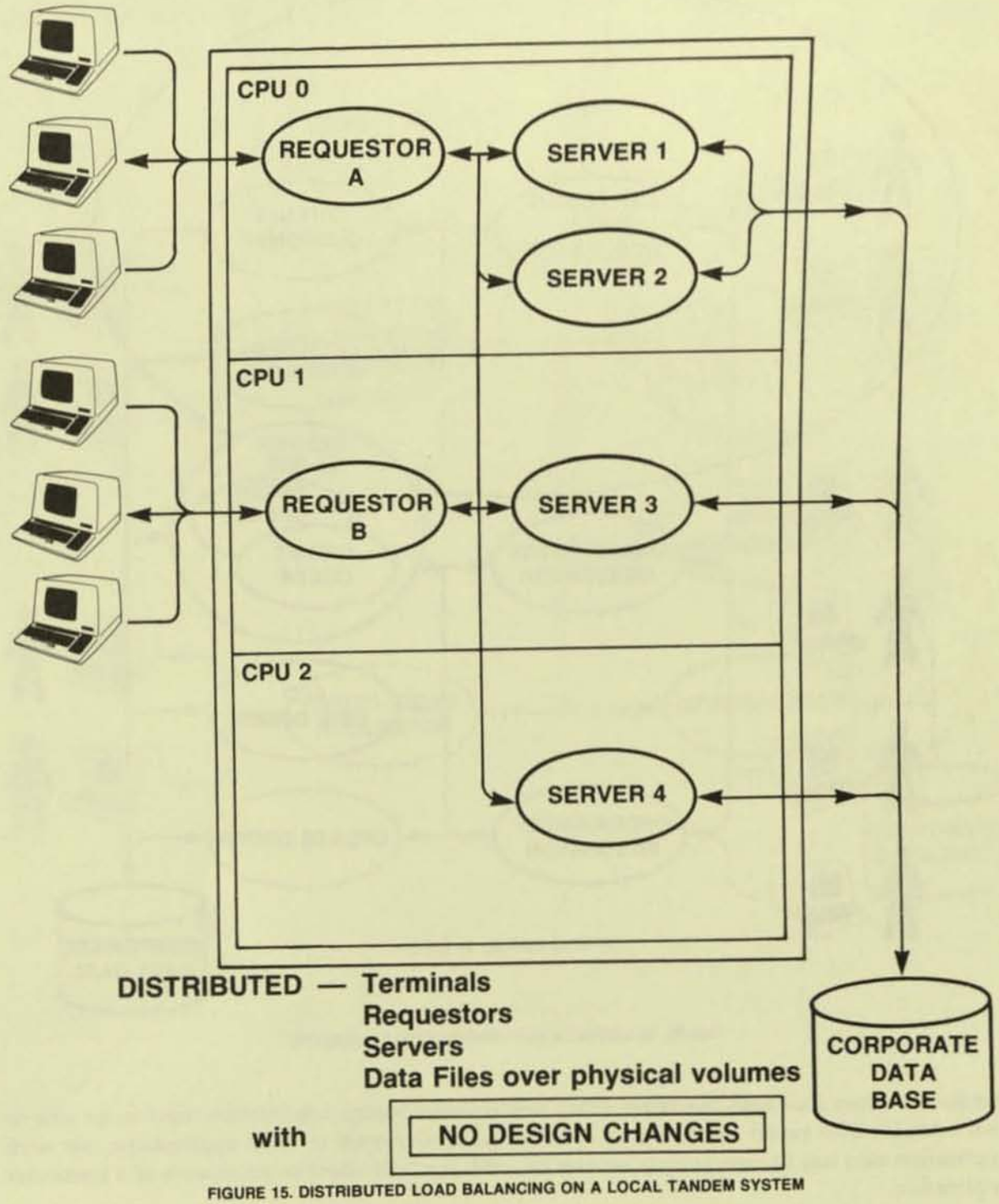

FIGURE 15. DISTRIBUTED LOAD BALANCING ON A LOCAL TANDEM SYSTEM

The next step in distributing application functions is over a network. The mechanism for communicating between processes within the TANDEM environment is consistent (Read / Write). Therefore, distributing application functions over a TANDEM network using EXPAND does not impact the original design. Figure 16 illustrates the distribution of application functions over a simple EXPAND network.

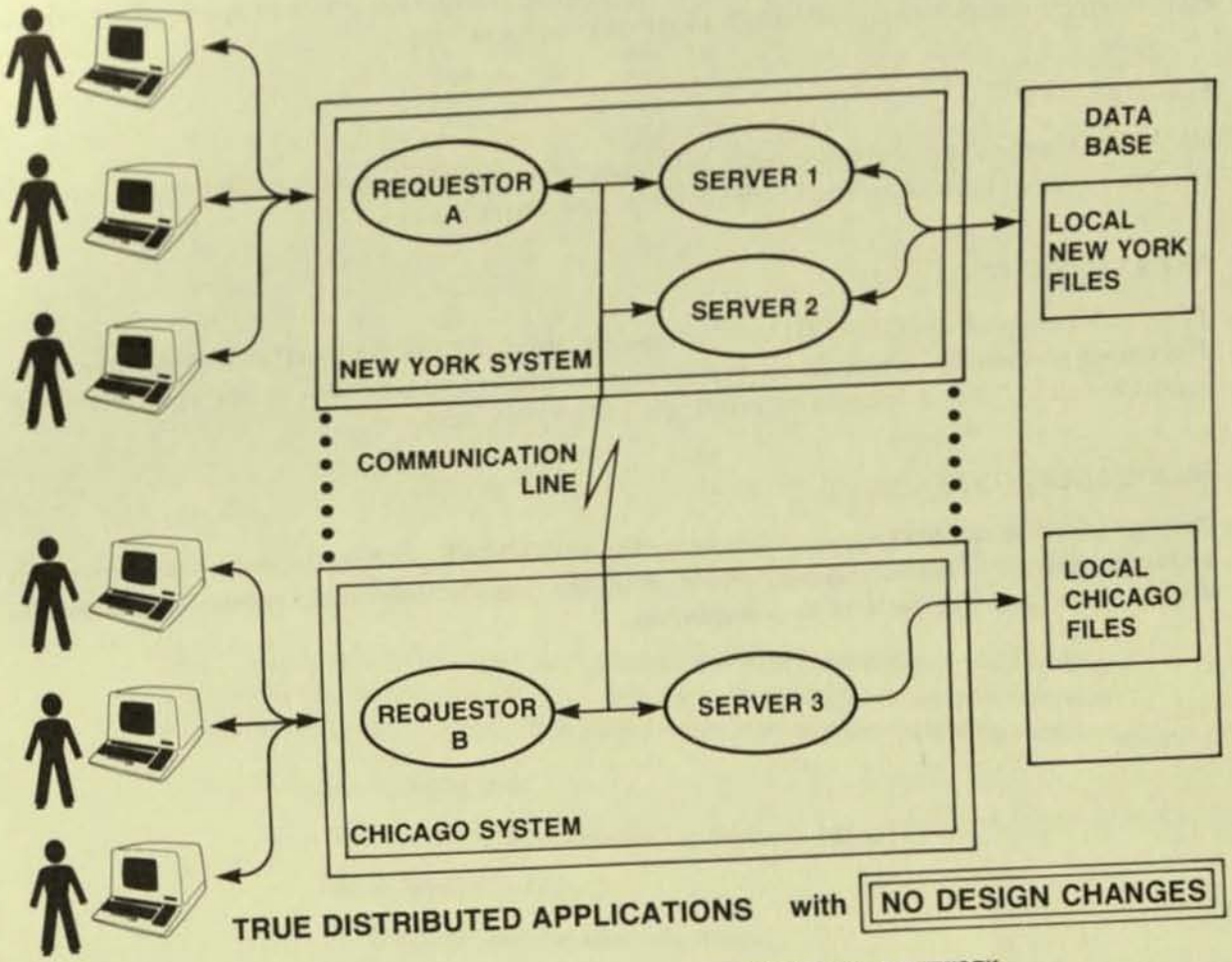

FIGURE 16. DISTRIBUTED LOAD BALANCING ON A SIMPLE EXPAND NETWORK

In summary TANDEM offers a total solution to transaction processing:

RELIABILITY:

NonStop transaction processing assures continuous system availability and data integrity

INSTALLABILITY:

PATHWAY offers a quick and easy way of developing transaction oriented applications, which significantly reduces the cost of applications development.

FLEXIBILITY:

PATHWAY offers the ability to make on-line additions, modifications, or deletions of transaction types, screen characteristics, applications, and terminals.

EXPANDABILITY:

The combination of the GUARDIAN Operating system and the EXPAND network offers true distributed processing, which allows applications to run in any processor in any system without regard for the physical location of terminals or the data base.

MAINTAINABILITY:

Because of the structured approach taken by the PATHWAY product, modules may be written and implemented in small, single threaded, and easy to understand components. Therefore, the maintenance task will be kept to a minimum.