Copyright © 1982 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 9, November 1982
Printed in U.S.A.

Database Systems:

Design of a Prototype Videotex System

By. N. H. GOGUEN

(Manuscript received October 20, 1981)

A prototype videotex system was developed for a concept trial. The system was based upon existing white-page and yellow-page services in an electronic medium. This paper describes the architecture for the message-driven, distributed system with functional specialization, which was developed to meet the prototype system requirements.

I. INTRODUCTION

This paper describes a prototype interactive videotex system built for a concept trial in which home and small business users would access a variety of services based on existing white- and yellow-page listing services using a modified television equipped with a full alphanumeric keyboard. As the term interactive videotex connotes, the user would have available a two-way information-delivery system providing selective access to electronic information transmitted over telephone lines from a computer system. The concept trial, and the system supporting it, would provide an experimental test-bed for determining user reaction to the medium and to the services provided in that medium. Independent waves of users would have access to the system over the trial period, and services offered could be added, modified, or removed from wave to wave based on user feedback in preceding waves.

The system design for the prototype had to accommodate several requirements:

(*i*) The system must be flexible to provide for growth in the services offered over the trial period. This growth could include the addition of new services with significantly different functional requirements, including transaction services utilizing database-management capabilities and personalized user services for the creation, storage, and access of individual user files.

(*ii*) The system must support dynamic tuning and reconfiguration. Since the services provided in the concept trial were not typically available in the home user market, no historical data on usage patterns and transaction mixes existed on which to engineer the system. Further, as users gained familiarity with system services, usage patterns were anticipated to change. In addition, based on the set of services available within a wave, usage from wave to wave could vary significantly. Dynamic tuning and reconfiguration capabilities would provide the means for allocating system resources based on actual usage. Reconfiguration capabilities would also be used to provide degraded service in the event of hardware failure.

(*iii*) Most importantly, changes in the services being offered must be made in a timely fashion. The system must facilitate rapid modification of services to provide maximum user response data on service preferences during the trial period.

(*iv*) A flexible database access capability must be provided to allow users to access a database of one million telephone listings by a variety of inputs. To meet schedule requirements, an existing prototype listing retrieval database system for use by telephone operators, which ran under the *UNIX** operating system on a minicomputer, would be enhanced to provide this capability.

II. APPROACH

The initial design phase benefited from the existence of an earlier prototype in a laboratory environment on which performance studies were conducted as input to a system model. The studies were made under the *UNIX* operating system on a minicomputer and included the implementation of telephone-listing retrieval services that utilized the usual *UNIX* process structure—a tree of processes created via "fork-exec." That is, a new process was created and executed when a customer process requested a service and terminated when the customer process no longer needed the service. In analyzing the resource requirements and performance characterized by the system model, the need was demonstrated for more efficient interprocess communication facilities beyond the standard *UNIX* "fork-exec" to meet the prototype system service requirements.

To satisfy the system requirements and to provide the needed extended interprocess communication capabilities, a system design

* Trademark of Bell Laboratories.

based on a message-driven process architecture was adopted. Services were implemented as sets of distinct, cooperating processes that communicated via message passing. The *UNIX* operating system was augmented with a device driver called the distributor, which provided an interprocess/interprocessor store-and-forward message-passing facility among unrelated processes.

To ensure adequate resources for the initial set of services, reconfiguration resources in the event of processor failures, and a growth path in the event that significant new services and/or additional users were added in the trial period, the system was implemented as a local network of minicomputers interconnected with a high-speed bus. The distributor implementation supported message passing among unrelated processes on a single processor and with processes connected to distributors on other processors.

The message-driven process architecture provided a disciplined approach for the design and implementation of the services. Each process was designed, wherever possible, to provide a generalized function that could be used by more than one service in the system, as for example, user input validation. Each process was data driven and provided a well-defined set of tasks. In addition, the message-driven process-architecture approach provided the application developer with a uniform view of distributed resources. The developer did not have to program differently based on process location. Explicit awareness of resource location could be taken into account as needed for performance considerations, but the message passing access to resources allowed hiding the explicit location level of detail in the higher levels of abstraction in the program design. This design approach facilitated timely implementation of services.

The network implementation provided the opportunity to functionally specialize one or all of the minicomputers. This enabled us to provide efficient use of resources at the possible expense of the loss of flexibility in reconfiguring in the event of processor failure. Based on the performance and modelling studies conducted on the early instance of the prototype telephone-listing retrieval service implementation, one of the minicomputers in the network was configured as a backend database machine. The studies had shown that telephone-listing retrieval requests were characterized by a small amount of input data, a variable but larger amount of output data, and a CPU-intensive resource utilization. Further, performance measurements on the high-speed bus indicated that the response-time overheads for the communication-link transmissions would be minimal in the overall response time for processing a user request.

The remaining sections of the paper examine the approach in greater detail, including the hardware and software used in the system. The

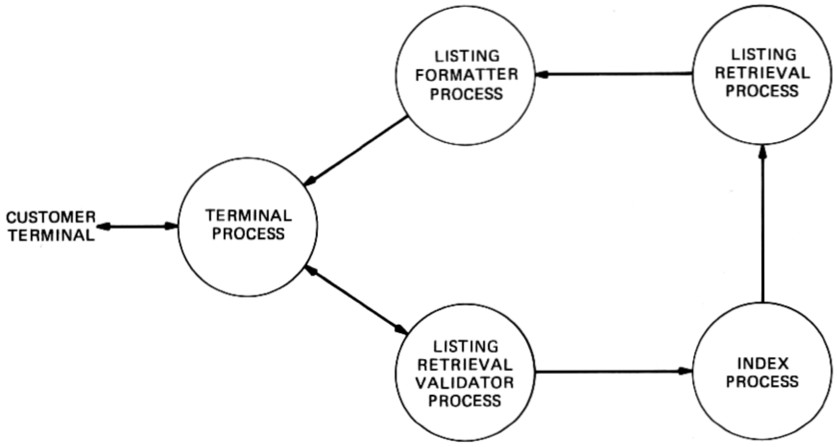
LISTING FORMATTER PROCESS
LISTING RETRIEVAL PROCESS
CUSTOMER TERMINAL
TERMINAL PROCESS
LISTING RETRIEVAL VALIDATOR PROCESS
INDEX PROCESS

Fig. 1—Listing retrieval process structure.

final section summarizes what we learned in the implementation process.

III. SYSTEM STRUCTURE

Logically, the prototype system services can be viewed as a collection of independent processes running on various computers in the local network and communicating via messages. A user service available on the system, such as telephone-listing retrieval, is provided by a set of cooperating server processes with well-defined tasks that process a given user request. The application programs that respond to the user request are data-driven at the process level, with messages being passed among the set of server processes that handle the request.

The process structure shown in Fig. 1 illustrates the message flow corresponding to a user request for a telephone-listing retrieval. The terminal process receives all user input from, and provides all user output to, a specific user terminal. The listing retrieval validator process receives requests to validate all user input for listing retrieval requests. If the user inputs an invalid request, the listing retrieval validator process sends messages to the terminal process for display to the user to allow correction. At the point when a valid request is made, the listing retrieval validator process sends to the index process a message specifying an expression consisting of the values of fields in listing records. The index process determines which primary and secondary index data can be used to select the records that satisfy the user request by parsing the selection expression. The index process produces a list of keys of candidate records that meet the request. The index process sends a message containing the list of candidate keys to the listing retrieval process which then retrieves from physical storage the records associated with the keys.

Any further processing required by the user request is handled by the listing retrieval process. The remaining records are sent as a message by the listing retrieval process to the listing formatter process for further application processing, including the formatting of the listings for display on the user terminal. The listing formatter process sends a message to the terminal process to display the formatted listings to the user. In fact, each server process described provides considerably more functionality than mentioned in the example.

In the message-driven process architecture, increasing the functionality of server processes to handle additional feature specifications was made easy. New message types were defined for the functions added, and the sender-receiver message communication was extended to include the additional types. Similarly, server processes that supported new classes of functional capabilities were incorporated. In the listing retrieval service, incorporation of a new capability to provide color/graphics advertisements associated with business listings required (*i*) increasing the functionality of the listing retrieval validator process to handle validation of requests for advertising, (*ii*) adding advertising fields to the listing records containing pointers to a database of ads, (*iii*) building additional indices on the advertising fields, (*iv*) increasing the functionality of the listing formatter process to recognize and display advertising codes, and (*v*) increasing the functionality of the terminal process to permit the display of advertising associated with business listings returned in response to the user request. In addition, an ad interface server process was implemented to control the advertising database requests. A new message type was defined for communication between the terminal process and the ad interface process.

IV. COMMUNICATIONS SYSTEM

The mechanism for providing message passing was the distributor device driver extension to the *UNIX* operating system. The distributor is a store and forward message processor that allows independent processes to communicate with each other. A process sends a message to the distributor for delivery to a receiver process. The distributor queues the message for the receiver process in its local memory until the receiver process is ready to accept it. No blocking of the sender or receiver process is involved, so that both the sender process and the receiver process can continue to operate in parallel. However, the distributor must provide message buffering capabilities and manage buffer pools. The solution to message buffer pool management is an upper bound on the number of messages that can be on a queue, with that upper bound tunable by the system administrator.

The distributor is configured with a number of independent channels for use. Channel initialization defines how the channel can be accessed by sender processes and what characteristics pertain to receiver processes for outgoing messages. One of the channels has special characteristics, including the ability to clear an internal statistics buffer maintained by the distributor, and is known as the control channel. In the prototype system, a control process is attached to the first available channel and is used to monitor system status and to supervise the distributor.

At system initialization, a startup function reads a command file that includes information on the names of all server processes associated with each subsystem of the prototype system being initialized. Optional run-time parameters associated with each process can be included in the command file, as, for example, a host option specifying the host processor on which the process should be initialized. The commands are parsed and placed in a message buffer, with run-time-option information placed in flags included in the messages. The messages are sent via the distributor to the control process on the machine specified by the host option, with each process listed in the command file being sequentially executed.

Each process that communicates through the distributor must communicate through a unique distributor channel. The control process dynamically allocates distributor channels to processes at run-time and makes these channel identifiers globally available in a data area called the environment. In addition to distributor channel information, the environment contains parameter data common to all processes in the system.

Terminal interface processes are created and destroyed dynamically when a user logs in and out of the system. At log-in, the startup function opens and sets a distributor channel, forks and executes a terminal interface process, sends a notification message to the control process, and exits.

The control process maintains a route-table of status information for processes on the distributor. In the prototype system, services are provided in a multiprocessor environment. A distributor and a control process are on each processor in the network. Interhost communication is managed by the control processes. Since server processes are addressed by name, the actual address of a process that provides that service is associated with the name in the route-table entry for that name. The address contains a host identifier that specifies which host the server function is on. Control processes on all active processors in the network communicate via a set of message types to keep the route-table on the processors accurate and consistent. The message types handle creation, deletion, and modification of entries in the route-table.

Several other capabilities of the distributor were used extensively in the prototype system. The first is the concept of a function group. A function group is a pool of server processes, each member of which provides the same set of tasks. When a process sends a message to a function group, the distributor places the message on the input queue of the function group member with the shortest input queue.

The function group capability provides flexibility in starting up the number of processes required to establish a particular set of tasks in a system. This creates a powerful tuning mechanism based on actual demand for a particular set of tasks and on availability of overall system resources. Based on system-usage characteristics, the number of members of a function group are adjusted either at system initialization time or dynamically.

Tuning based on system-usage characteristics over a period of time is useful in a mature system with a stable set of features. In an environment where the features may change rapidly because of evolving customer demands and where the customer's actual usage patterns vary, the ability to dynamically modify the number of servers available is needed. In this case, the control process monitors the queue lengths of members of a function group in conjunction with the overall demand for system resources. If queue lengths for all members of a function group are long, additional members are started up based on availability of system resources. Similarly, if queue lengths are small, some members of a function group can be shut down to free system resources for other service demands.

The system was also designed to allow a sender process to communicate with a specific member of a function group. This capability is used in cases where a message exceeds the standard message size and is broken into a first message and a series of continuation messages. When the first message is sent, the sender process requests the return of the channel identifier of the member of the function group which received the message. The continuation messages are then sent to the same member of the function group by specifying that channel id.

Another capability of the distributor is the availability of two special channels to which messages cannot be directly sent. The first, the sink channel, is used in the prototype system for system error logging. This channel is the only one that allows multiple opens. A process which does not wish to receive messages can open the sink channel and send messages to processes through the normal mechanisms. The second channel, called the "abandoned channel," is used to accept messages that are being abandoned. This channel can be used as part of a recovery mechanism for abnormal termination situations. On abnormal termination, if messages are present on an input queue for a process when the process closes its channel, the messages are lost unless the abandoned channel is open. Messages from the abandoned channel have two message headers which identify the message source and the intended receiver.

In this communication system, messages sent to a process are queued first in, first out, with no priority classes available. In this environment, signals are used to support interprocess exception handling. Exceptional events include power failure, death of a child process, untrappable kill, and quit (used for emergency shutdown). Signals are used to implement controlled shutdown.

V. MESSAGE FORMAT

A message is composed of three components: a distributor header, a system header, and message text. The two headers are fixed size and the message text variable size. The distributor header contains information for use by distributor routines, such as host origination or destination, channel identifier, and subdevice identifier. The system header contains information of global interest to the application processes. This includes the originating source of the transaction, timing information associated with a transaction, message modifier flags, a message type indicator, and a count of the number of bytes in the message text. The message modifier flags are used to indicate that an acknowledgment is requested on receipt of the message, that this is an acknowledgment of a message, and that this transaction continues. Message text is specific to the sender and receiver processes.

Variable-length messages are required since the text portion of a message can include as contents such items as all the telephone listings that matched a customer's listing retrieval request. In a single processor environment, a file could be used to pass large amounts of data between processes. In general, since the prototype system was implemented in a multiprocessor environment, all interprocess communication has been via messages, which avoids the interprocessor file-handling problem.

VI. BACKEND DATABASE MACHINE

One of the processors in the network is dedicated to providing a backend database function for the telephone-listing retrieval service to improve the overall system throughput and performance. In the prototype system, a general-purpose minicomputer is tuned for database functions. Since the operating system is limited in kernel address space, only those drivers required for the backend functions are included. Correspondingly, the host machine is configured without the drivers required for the data-management functions, and system resources are tuned for application functions.

Figure 2 depicts the host-backend connection in the prototype system, together with the placement of the application processes executed by the host machine and the database-management processes executed by the backend machine. Note that the scenario described in Section II for processing a customer request for a telephone-listing retrieval applies here. Since the concept of the backend Database Management System (DBMS) is simply a special case of a computer network, the communications system required to support the concept is available in the distributor implementation described in Section II. Host and backend interface functions are provided by the distributor. A message containing the database request is sent from the listing retrieval validator process to the distributor on the host destined for the index process. The address of the index process in the route table indicates that the host for the listing formatter process is the backend machine. The message is sent across the interprocessor link to the distributor on the backend machine that places it on the queue of the index process. Processing of the database request involves both the index process and the listing retrieval process. The data and status information returned as the result are sent as a message from the listing retrieval process to the distributor on the backend machine for delivery to the listing formatter process. The route table indicates that the host for the listing formatter process is the host machine. The message is sent across the interprocessor link to the distributor on the host machine which places it on the queue of the listing formatter process.

The database management modules operating on the backend processor are multiprogrammed. Function groups are used for the index process and listing retrieval process, with the number of members in each function group initially established to handle the projected telephone-listing retrieval-request traffic, with tuning as appropriate. The database tasks originated from user requests received on a host processor that were transmitted through the distributor and the communication system to the database-management modules.

Because telephone-listing retrieval services were the major set of services available in the trial, a heavy demand for such services was anticipated. By distributing the application programs and the database-management modules across two processors, overall throughput capacity was increased by allowing parallel execution of application programs, database-management modules, and disk accesses. This throughput capacity increase was at the expense of slightly increased total response time to the user based on transmission time across the

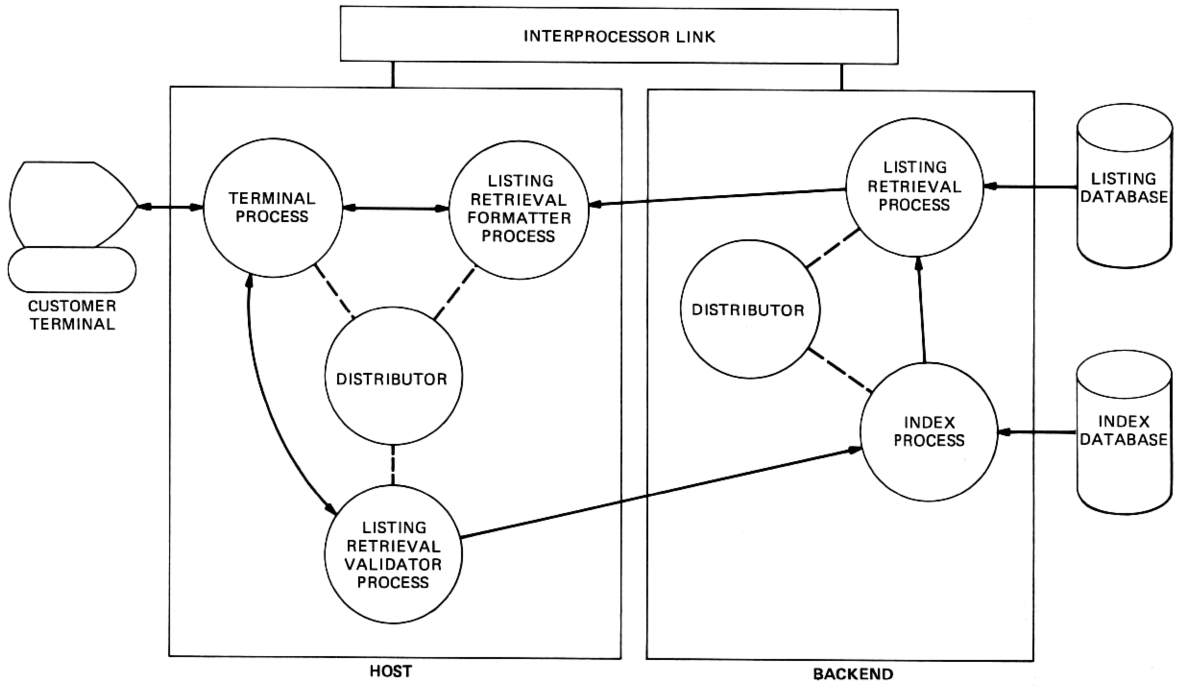

Fig. 2—Backend database management system in the prototype system.

high-speed bus and additional overhead in the communications subsystem. Since the modelling studies had shown that database access was the major system bottleneck, and since trial schedules did not permit moving to a new processor line with increased capacity, the specialization of one of the processors in the network to a backend database machine provided the throughput required.

Cost is another consideration in using a backend database machine. One factor in cost is the communication and interface software required to implement this alternative. In our case, the local network architecture selected included communication and interface software which could be used. Since all the processors were in the same family, secondary storage media compatibility and conversion overheads associated with the incompatible character sets, data formats, etc., were not issues.

VII. HARDWARE AND SOFTWARE ENVIRONMENT

The prototype system was installed on a local network of minicomputers linked together using an interprocessor bus with a 1-Mbyte-per-second data-transfer rate.

The system ran under the *UNIX* operating system, augmented with a number of device drivers used to extend the core operating system capabilities to support the application requirements. Driver requirements occurred in several areas of the system. A terminal driver was written to support color/graphics user and frame-creation terminals. These were block mode, asynchronous terminals which used an X3.28 protocol. Another driver was written to support database requirements for semaphores used in locking.

The network-control functions and message-driven architecture were made possible by development of several drivers: the distributor, a driver to support the interprocessor link, and a driver to support the interface of the distributor to the interprocessor link.

An in-memory files driver was written to provide a shared data capability for processes that were guaranteed to run on the same processor based on performance considerations. This driver was replaced with the shared memory capability when version 4.0 of the *UNIX* operating system became available. The application was written in C language.

VIII. CONCLUSIONS

The original requirements specified for the prototype system were met in the course of its design and implementation. New services were proposed as, for example, the advertising services associated with business telephone-listing retrieval, and these were designed and incorporated into the system in timely fashion. The message passing process architecture, in which distributed resources could be viewed as logically centralized, aided the rapid and disciplined development of new server functions. New services, or service-providing functions, could be added, deleted, or relocated by modifying a systemwide route table, without requiring changes in program logic or in resource access mechanisms.

Using the function group capability and the monitoring of process queues, the overall system configuration could be tuned to meet the actual processing requirements in a particular period. The availability of processor capacity in the local network provided hardware for reconfiguration in degraded mode in the event of hardware failure. The route-table definition of the location of processes provided the logical basis for reconfiguration.

The specialization of one of the processors to provide backend database-management functions allowed us to provide acceptable throughput using the existing telephone-listing retrieval components, modified to run in the message-passing process architecture, and to provide new functions required for the prototype system services.

IX. ACKNOWLEDGMENTS

The design and development of the prototype system has been a team effort of many people over a few years. The author wishes to acknowledge the contributions of all the people whose work is summarized here. The original design of the distributor driver was done by Marc Pucci and extended for the prototype system by Don Cragun. The pioneering work in the area of backend database machines was done at Bell Laboratories by Rudd Canaday and his associates.