Edgar H. Sibley
Panel Editor

Building and prototyping an agricultural electronic marketing system involved experimenting with distributed synchronization, atomic activity, and commit protocols and recovery algorithms.

THE DESIGN AND BUILDING OF ENCHÈRE, A DISTRIBUTED ELECTRONIC MARKETING SYSTEM

JEAN-PIERRE BANÂTRE, MICHEL BANÂTRE, GUY LAPALME, and FLORIMOND PLOYETTE

Recently, there has been growing interest in electronic marketing systems, particularly for agricultural products [10, 16]. According to Lindeman Schlei, an electronic marketing system provides

> a very efficient price discovery mechanism; it also separates the distinct functions of negotiating a trade and physically transferring a product by centralizing negotiations and decentralizing product transfer. . . . Its success depends on: reliable technology, high trading volume, reliable grades and standards for descriptive selling and reasonable charges [16].

In this article, we describe the history and the development of *Enchère* (meaning "auction" in French), a distributed electronic marketing system that satisfies the criteria of speed, reliability, and fairness between buyers and sellers as required by the market. To achieve this, Enchère combines many original features that up to now have been applied mainly to separate research products: Enchère is a distributed system consisting of a loose network of autonomous microcomputer-based workstations communicating with each other via messages. Reliability is guaranteed by means of atomic actions that are supported by new hardware and software products. New stable memory boards were developed to guarantee the reliability of transactions implemented by dynamic process creations.

This work has been partially supported by the Agence pour le Developpement de l'Information (ADI) (Information Development Agency) and by the Societé de Fabrication et de Réalisation Electronique (SOFREL) 35230-Vern-sur-Seiche-France.

In this article, we examine the workings of a centralized auction bidding system—extracting fundamental operating rules and principles and isolating the requirements of a new electronic system. We then report on the proposed software and hardware structures—defined *in that order*—and their implementation in two steps: a working prototype run on a big system (MULTICS), and a second preindustrial system built from the first without modification of the original software ideas.

THE ELECTRONIC MARKETING PROBLEM AND OUR SOLUTION

Electronic Sales

In Europe, agricultural products are traditionally auctioned by means of a system known as the "dutch clock." In the dutch-clock auction, all buyers gather in one room where they are presented product lots for bidding by a sales manager representing the sellers. In front of this room stands a big clock with numbers around it representing possible prices for the lots (e.g., figures ranging from $2.00 to $0.50 in decrements of $0.05); the sales manager positions the arm of the clock at the starting price for the lot, and the arm descends regularly until one of the buyers in the room stops it by pushing a button in front of him or her. An electronic system identifies the person who responded first and reports this buyer to the sales manager who now decides to accept the offer or not. The pace is a fast one:

The sale of a lot takes about 10 seconds. The dutch-clock system is well accepted by the sellers and buyers, but it requires that all participants be centralized in one place.

In the system in Brittany, France, which served as the inspiration of our work, three such rooms are already linked so that buyers in one room can bid on the lots presented in all three rooms. This interconnection helps minimize transportation costs, lessens variations between nearby sale rooms, and permits buyers a better choice and sellers a greater number of prospective buyers as well as an expanded market. In systems like the Brittany auction, the order of presentation of the lots is crucial because prices vary considerably depending on the time elapsed from the start of the sales: For a variety of reasons, prices usually start low and go up in the middle of the sale and down at the end. To ensure fairness, the system permutes the lots of each seller daily; this causes no problem in the Brittany system because all sellers are members of the same organization that is responsible for setting up the auction and for compensating sellers whose lots are presented first.

However, it is now necessary to expand the market to allow more than one seller organization. To achieve the same kind of "justice," the lots of each seller should be interleaved; however, this will lengthen an already considerable total sale time of three to four hours. The problem is how to reconcile those two contradictory goals of speed and market expansion.

Our solution relies on the following observation: Although each buyer might be interested in the lots of all sellers, he or she usually tends to satisfy his or her needs through only a few sellers (and often only one). Since the buyer tends to buy elsewhere only if he or she cannot find what he or she needs locally, we proposed the creation of logical sales rooms where each participant (buyer and seller) is given the means to deal with only the people he or she wishes. Each buyer uses a "workstation" consisting of a microcomputer linked to seller workstations through a combination of local and national networks. Each station implements algorithms that initiate communications between participants who want to deal with one another. The global system, which is implemented by local algorithms executed independently on each microcomputer, is so structured that transactions involving disjoint groups of buyers and sellers can proceed in parallel.

In this scenario, a buyer or seller uses a workstation consisting of a microprocessor-based control unit with a small display, a dutch clock, and keys that allow him or her to participate in the sales. There is also an ordinary screen display showing the results of all transactions being processed. Although the hardware for each user is the same, the software and logical meaning of each key are different depending on whether the user is the seller or buyer. A physical description of the system is given in [2]; here we will discuss only its operational aspects, beginning with the buyer's version.

The buyer chooses the sellers he or she wants to deal with and may change his or her choice at any time. The proposition from a seller is displayed on the buyers control unit, and the clock shows the starting price for the lot as given by the seller; the clock registers the regular downward movement of the price. When the buyer decides a particular price is agreeable, he or she pushes a key to send the offer to the seller. The buyer then awaits an answer from the seller telling the buyer if he or she has gotten the lot or not. The buyer may then receive a new offer (possibly from another seller). This offer is chosen by the system from the set of sellers with whom this particular buyer is dealing.

The seller, of course, sees things rather differently. By pushing a key, the seller sends his or her proposition to interested buyers and waits for their offer. When all buyers have answered, the value of the best offer is displayed on the seller's control unit; the seller decides either to accept or reject this offer and transmits this decision to the buyer. The seller may now send his or her next proposition.

Application Requirements

By its very nature, the electronic marketing context imposes unique demands on its computer implementation in terms of the order of sales, fairness, reliability, extensibility, speed, and distribution.

Electronic marketing requires that sales initiated by a seller are carried out sequentially. The concept of fairness between buyers and sellers requires that all buyers have equal opportunity to buy lots proposed by sellers, and that sellers have an equal chance of having their proposition seen at any given time by different buyers, there being no a priori order established among sellers. This fairness doctrine differs somewhat from that commonly referred to in the distributed-systems literature in the sense that it must take into account competition between sellers and not only the absence of "starvation"—the failure to take into account lots of a given seller due to unfair choices by the system. A crucial doctrine in the context of commercial marketing, fairness must be enforced in all cases—including defective functioning. However, it cannot be achieved at the expense of real-time efficiency measured in terms of availability and speed of processing. The fairness doctrine requires that seller lots be equitably interlaced and not indefinitely delayed, and that buyers make their decisions based on the same available information.

Ensuring reliability means that, at each session, Enchère must provide complete service throughout the session (several hours a day). Should a failure occur, buyers and sellers must be able to continue the sales, even though transaction time may be lengthened.

Another requirement is that the system architecture be extensible: It should permit the easy insertion and removal of buyers and sellers without changing the overall architecture. The question of speed is also critical because all current systems are acknowledged to be unacceptably slow due to deficient computing power. Finally, an electronic auction system of this kind is by definition a distributed one: Users may be spread over a

large geographical area encompassing a region, a country, or even several countries.

User Requirements

An electronic marketing system is used by sellers and buyers cooperating (or competing) to negotiate the exchange of goods. In this exchange, the following considerations are critical:

- *Ease of use.* Since buyers and sellers are not computer specialists, operating commands must be as simple as possible and yet powerful enough to conclude a deal with dispatch (i.e., in 10 seconds).
- *Confidentiality.* The system must guarantee a certain confidentiality with respect to buyer–seller business dealings. Specifically, a buyer must not know the bids of other buyers except for the price of the highest, which is broadcast to everyone. A seller, on the other hand, must receive only the bids relating to his or her sales and not the bids made by buyers dealing with other sellers. The system must also ensure that a user can neither block another user nor take the place of one.
- *Autonomy.* Every user should be independent in the sense of being able to join or leave the service as he or she wishes.
- *Availability of personal computing power.* Since the auctions generally take only two or three hours per day, the system should be usable for other purposes during the remaining hours.

ARCHITECTURAL DESIGN

To build an architecture for Enchère that respects the autonomy and independence of each user while providing acceptable performance, two possibilities were studied: (1) a multiprocessor architecture on a central site that could be accessed via specialized terminals, and (2) a distributed architecture where each user would have a personal computer and Enchère would be the set of all those computers communicating with each other to complete a sale.

In a centralized system, achieving user autonomy and flexibility (e.g., users' accessing or leaving the system at will, and buyers' choosing the sellers they want to trade with) is difficult because all users are dependent on the central site and unwanted dependencies are created between users. A seller, for example, might depend on a particular buyer, or vice versa, even if the one is not particularly interested in trading with the other. To solve those problems, we defined the Enchère session as a set of autonomous units [4] linked by a communication network, where the implementation is based not on one site but as many sites as there are users at any given time.

This means that the malfunctioning of a site affects only the users depending on it for their transactions. It also means that users can work separately: Many sale sessions can proceed in parallel as long as their set of users is disjoint.

Logical Structure of the Application

The implementation of each sale is modeled around the notion of an "activity" [1, 5]—composed of a set of cooperating processes (one at each site)—so that the overall structure of Enchère at any time becomes a "tree" of activities.

The initial activity involves an opening process at each user site whereby the user signals a willingness to participate in the sale. By this process, each user is made known to the others, and communication links to processes are established for dealing explicitly with the sale. The user-site processes join or leave this activity dynamically: A user uninterested in a sale may leave while the others continue without him or her. When all users have left the network, this activity ceases.

For each lot offered for sale, dependent activities are created that involve an instance of a seller process and instances of each buyer process interested in the sale. In the following configuration of buyers and sellers,

three sellers sites—SS1, SS2, SS3, and
six buyers sites—BS1 to BS6,

dependencies between buyers and sellers are defined by the sets of buyers SB(· · ·) interested in a given seller. Given the following sets of interested buyers and sellers,

$$SB(SS1) = \{BS1, BS2, BS3\};$$

$$SB(SS2) = \{BS4, BS5\};$$

$$SB(SS3) = \{BS1, BS5, BS6\}.$$

The buyers interested in making transactions with SS2 are BS4 and BS5. Should all three sellers initiate a sale, an activity structure results (Figure 1, p. 22), wherein PBij is the ith instance of a process representing the behavior of buyer BSj and belonging to the ith activity, and PSij is the ith instance of a process representing the behavior of seller SSj. At the BS1 buyer site, there is a buyer process involved in an activity for the sale of a lot from SS1, and another dealing with SS3.

By virtue of this activity scheme, all the dynamic properties of the application (i.e., independence, asynchronism, and competition for CPU time or memory management of the processes) are taken care of by the operating system and are of no concern to the application programmer. With this scheme in place, the sale scenario becomes very simple, as can be seen from the Ada® program given in the Appendix.

These sale processes terminate as soon as a sale has been completed. Note that a buyer is notified of the rejection of his or her offer as soon as a better one is received by the seller; the buyer may then participate in another sale because he or she is sure he or she will not acquire that particular lot. The synchronization mechanism for this "optimization" is taken care of by the activity mechanism and does not have to be dealt with in the application program.

Another important characteristic of the sale process

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

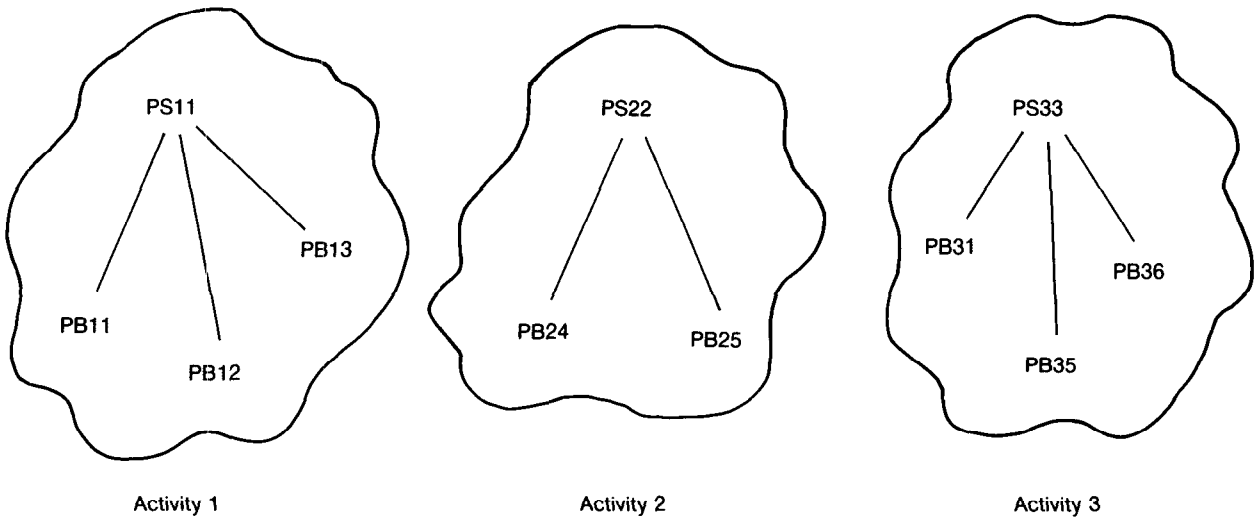

FIGURE 1. The Activity Structure

is the indivisibility associated with the execution of an activity. Two activities (father and son) only communicate by parameters on creation or by result on completion. The characteristics and consequences of this indivisibility and atomicity, which are akin to the transaction concept found in database systems [7], are described in detail in [2].

Physical Architecture of the Application

Ideally, every user is provided with his or her own workstation: Enchère is then implemented by a set of workstations interconnected through a communication system as shown in Figure 2, where SW means seller workstation, BW buyer workstation, and CS communication system.

In the current configuration, every user is provided with a personal terminal consisting of two devices: an application processor (AP), which manages the auctions, and a workstation (WS), which is the Enchère terminal. The WS is used for accessing the Enchère service, and every WS is connected to an AP.

Application Processors. Enchère's APs are built from two microprocessors (I8086 and I8085), which are connected via Intel Multibus® and use disk storage. A stable storage (discussed in more detail in Design and Implementation of Stable Storage, p. 26) is also connected to the I8086 through Multibus.

The I8086 processor manages the auctions and the local peripherals, while the I8085 acts as front-end processor for the I8086, controlling communication to and from the workstations and the communication system (Figure 3). A common memory facility is provided in addition to the stable storage mentioned above.

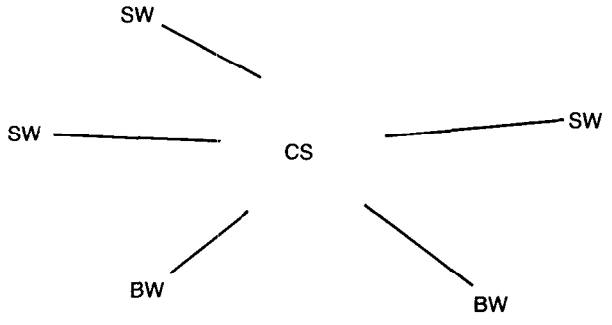

FIGURE 2. Enchère Architecture

The Workstation. The workstation consists of the usual display terminal and specialized command terminals that are managed by a microprocessor (Figure 4). The command terminal provides the commands necessary for participating in a sale—new sale, offer, refusal, etc. Every such command is communicated via a special button pushed by the user. These command terminals are also equipped with a special display that shows instantly all displayable information concerning the current sale.

The Communication System. The application is not tied to any particular communication system in order to ensure that Enchère can be installed on any "machine" possessing what we call the minimum interface properties: These include a communication system that allows message exchanges with acknowledgment, and communication error detection and notification.

CONSTRUCTING A PROTOTYPE

Since the proposed system was new and innovative compared with existing models, it was difficult to foresee the reactions of potential users. Moreover, for various reasons such as credibility and funding, it was important to demonstrate in advance the external characteristics of the new system.

Multibus is a registered trademark of the Intel Corporation.

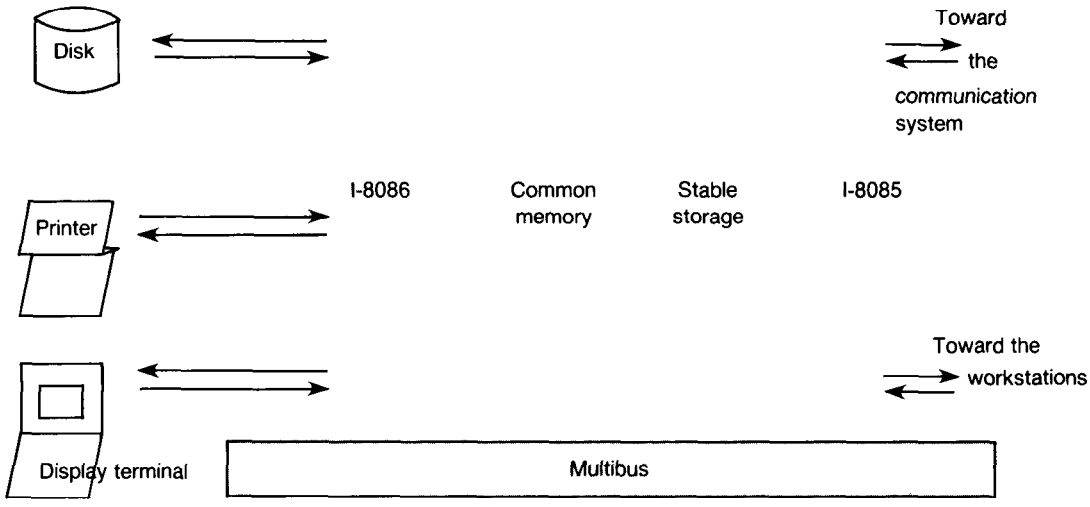

FIGURE 3. The Application Processor

With these considerations in mind, it was decided to approach the prototype in two steps. Constructing a first prototype on a big system (MULTICS in our case) would demonstrate the external features of Enchère to the eventual user, a nonspecialist in computing. This system would be centralized. Then, depending on user reception of the first prototype, a second prototype—a preindustrial system—would be built.

Since the budget for developing these prototypes was limited, some design decisions had to be made very early to ensure that the major part of the software produced for the first prototype would be reusable for the second. Specifically, it was decided that the final structure of the system would be "frozen" at the time of the first prototype. The second prototype would provide only an alternate implementation of the same abstract structure. Second, to limit code rewriting, the software tools used in the development of both prototypes were to be the same. This was achieved by virtue of a local language for microprocessor applications whose compiler produces intermediate code for a virtual machine, and then appropriate code for the actual target machine.

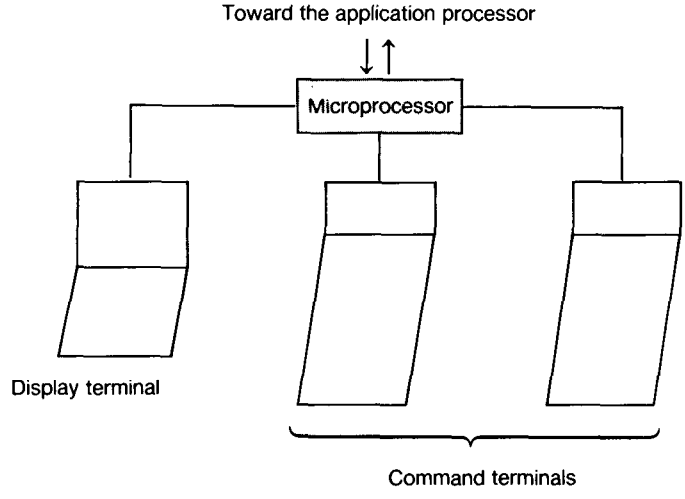

FIGURE 4. The Workstation

Implementing the First Prototype

The emulation of the Enchère activity structure was done on MULTICS, a process-oriented operating system [14]. Under MULTICS, when a user logs on, a process is created and associated with this particular terminal. This process can then create other noninteractive processes called "absentees," which in turn spark other processes and so on. The processes communicate via a system module called IPC (InterProcess Communication), which permits the sending of short control signals by means of unidirectional channels. Using these tools and a mutual exclusion mechanism, we built a communication system among instances of processes, the structure of which is shown in Figure 5. We also developed a probe system that allowed us to keep track of all

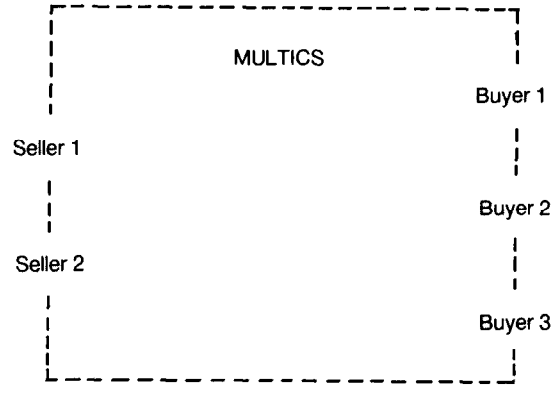

FIGURE 5. Architecture on the MULTICS System

simultaneous messages; this probe enabled us to simulate breakdowns and time-outs, and to evaluate the effects of software and hardware failures.

Users are linked via the MULTICS system. Each logical entity associated with a user incorporates many modules, a few of which are implemented on the workstation microcomputer and the others under MULTICS. Each of the buyer and seller boxes given in Figure 5 can be subdivided as shown in Figure 6.

As the software structure of Enchère is based on the notion of activity, its implementation under MULTICS also reflects this orientation. For example, in the sale of a lot from seller S to two buyers B1 and B2, each site is represented by a site process that interprets commands issued by the user. Thus, we have three site processes: SS for site S, and SB1 and SB2 for sites B1 and B2, respectively. When a new-sale command is issued on site S, process SS signifies that a new activity involving itself and SB1 and SB2 has to be initiated. This is achieved by creating an instance of seller process (PS) on site S and two instances (PB1 and PB2) of buyer process on sites B1 and B2. These processes are linked to form a new activity as summarized in Figure 7.

The buyer site, on the other hand, may create many instances of processes PB*i*, one for each independent sale (i.e., for sales activated by distinct sellers). Although costly, this approach has the advantage of simplicity and mirrors almost exactly the software structure decided upon.

Although this first prototype was important in the sense that it provided a working model of the system and user feedback, and also validated our structuring methods, it did not permit a fully satisfactory quantitative evaluation of the proposed service because, in a time-shared system, the number of external users can greatly influence the response time to commands.

This first experiment made clear that the original idea of having identical workstation keyboards for both sellers and buyers was not a good one in terms of ease of use: Too many functions were given to each participant, so for the subsequent experiment we designed two different keyboards.

The experiment also demonstrated the need for simplicity. In developing the first prototype, we should have restricted ourselves to a small demonstration unit, providing an external view of the system, and avoided putting in place the entire activity structure, which proved too expensive for use in real time. Although a test of the logical feasibility of the structure was important enough to warrant an experimental implementation, it should have been independent of the demonstration unit.

However, the demonstration was well received and prompted us (and those investing in the project) to con-

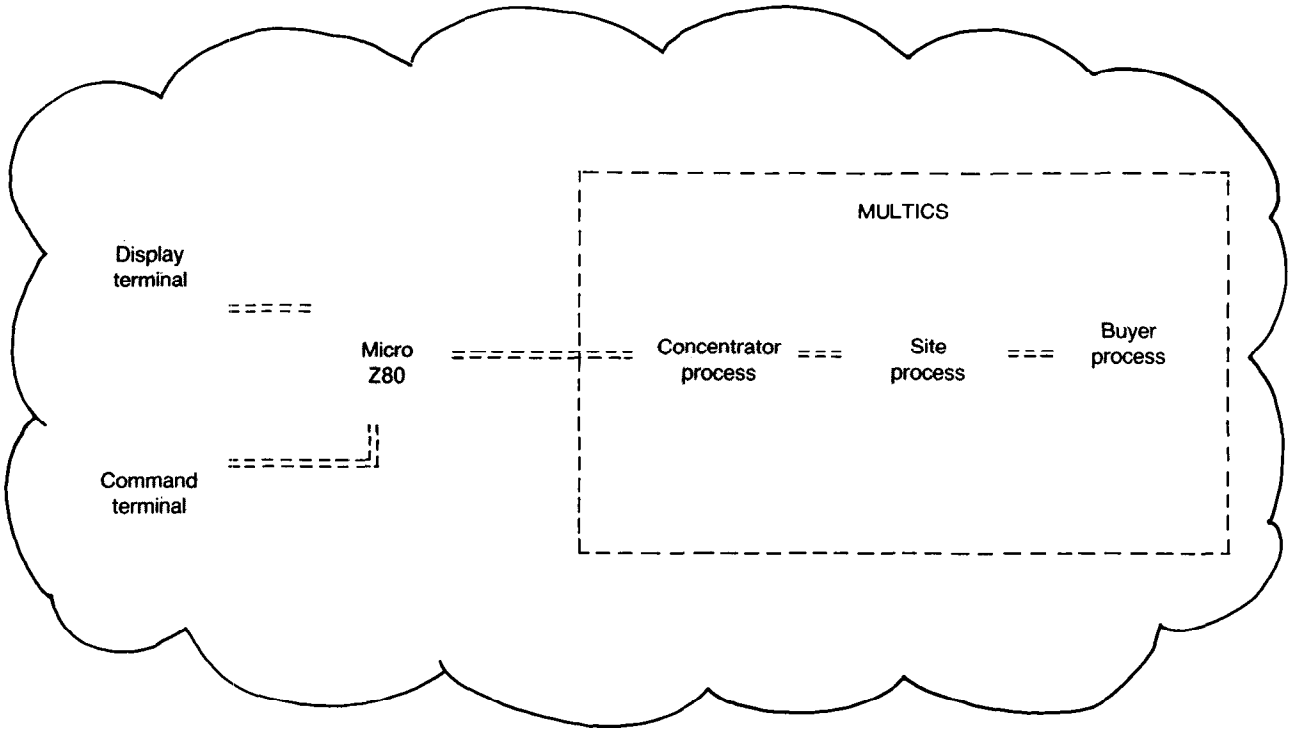

FIGURE 6. Representation of a Buyer

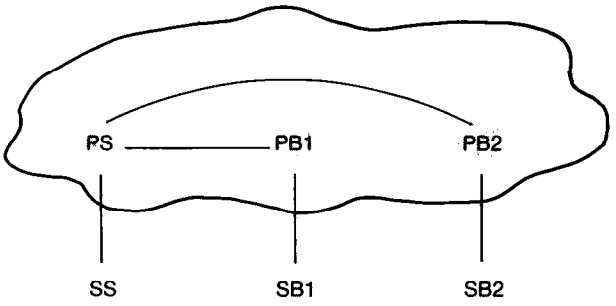

FIGURE 7. MULTICS Activity Implementation

tinue in the same vein: that is, to implement both site and sale processes on a network of microcomputers linked together first via MULTICS (which would then be confined to a "safe" network role) and ultimately via a combination of local and national networks.

THE SECOND PROTOTYPE

The second prototype consisted of three application processors and five workstations connected, in a first stage, through a concentrator (I8086). (Enchère is so designed that no problem should arise when it is transported onto a network.) In this section, we examine the second prototype as it incorporates the features of fairness and atomic transactions.

Fairness

Presenting propositions to various buyers in the same order for each seller ensures that buyers may all elaborate their strategy based on the same information. From a technical viewpoint, this is critical for avoiding deadlocks, since a buyer must terminate one sale before participating in another. Imagine the situation where two lots L1 and L2 are to be sold, L1 proposed by seller S1 and L2 by S2. Two buyers B1 and B2 compete for these lots. If B1 bids on L1 first (transaction T1) and B2 bids on L2 first (transaction T2), neither T1 nor T2 will ever be completed because in order to be completed T1 needs a bid from B2, who cannot provide it before completion of T2 (the usual deadlock situation).

To ensure that propositions are presented to buyers in the same order for each seller, a time stamp giving the (local real) time of emission and the name of the sender is appended to the message corresponding to the proposition of a seller. To choose the next proposition, the system always takes the message having the smallest time stamp. When the times are the same, an order of priority can be based on the names of the sellers. Lamport proposes a way to achieve this without necessarily defining a priori a total order on the names [8].

Ensuring that a seller has finished with a sale before starting another requires that the local choices (made by the computers on each buyer site) be coherent. That is, the computer must determine if the buyer has already received the "oldest" proposition before making the choice of propositions to be presented next. To do this, it is sufficient that each buyer-site B select at his or her local time H(B) the seller proposition possessing the smallest time stamp less than H(B) − DT. (DT is the maximum transmission delay between two sites.)

The above solution, however, is not sufficiently powerful to ensure that no priority is given by a buyer to the proposition of a particular seller, as can be seen in the following example:

S1, S2, and S3: seller sites;
B*i*: buyer site linked to S1, S2, and S3.
S1 starts a sale (s11) with his or her local time $h1 = 10$,
S2 starts a sale (s12) with his or her local time $h2 = 15$, and
S3 starts a sale (s13) with his or her local time $h3 = 20$.

The time-stamp solution instructs B*i* to choose s11. Assuming that the length of the sale is 3 units of time and DT is 1 unit of time, when s11 finishes, S1 initiates another transaction s11′, which is stamped ($h1' = 15$), and B*i* chooses s12. When s12 is finished, B*i* must choose between s11′ and s13. The time-stamp method forces B*i* to select s11′, thereby giving priority to S1, whereas S13 should have been chosen.

Instead, the buyer site B*i* (dealing with sales of many sellers) must be able to compare the times given by the local clocks of these seller sites: That is, if, at a given time H(B*i*), two sales' time stamps of sellers S*j* and S*k* are H(S*j*) and H(S*k*) such that for B*i* (H(S*j*) < H(S*k*)), the next sale started by S*j* will be stamped by $H(Sj)' > H(Sk)$. In our example, s11′ should have been dated by a value greater than 20.

To solve this problem, we introduce a notion of "fuzzy" time that is defined as follows: The local clocks of user sites U1, . . . , U*n* show the same fuzzy time as the local clock of a site U*i*, if and only if, $|H(Uj) - H(Ui)| < dt$. (dt is the maximal drift between the clocks of user sites U*j* and the one of U*i*.) Assuming the length of a transaction is greater than $2 \times dt$ and that all seller sites for a given buyer have the same fuzzy time, then fairness is fully guaranteed [2].

Cases where these conditions are not fulfilled have been studied carefully. In particular, algorithms are being reviewed that allow clock resynchronization following misfunctioning of a site.

ACHIEVING RELIABILITY

The Enchère system is designed to resist both software and hardware faults that may result in uncontrolled memory accesses. The crucial impact of uncontrolled memory access on the Enchère system was first revealed with the construction of the initial prototype. To ensure high reliability and avoid memory degradation, steps were taken in two main areas: implementing stable storage and instituting commit protocols and a recovery algorithm.

Design and Implementation of Stable Storage

Stable storage [9] is generally viewed as a memory device whose physical storage is stable (i.e., information does not degrade over time) and whose write operation is atomic. Usually, stable storage is built from disks, which, although they do not directly provide stable storage, do possess properties from which stable storage can be implemented. In Enchère, each application processor is equipped with stable storage that may contain two different types of objects: (1) objects of small size that are accessed directly by processes (e.g., variables) and whose lifetime is that of the activity that created them, and (2) objects that are stored for a long period of time (e.g., files). Since objects of type (1) cannot realistically be stored on devices like disks, as the time necessary to access them would be excessive, Enchère's stable storage is constructed of two devices: a stable RAM unit (SR) that is a part of the machine address space; and a disk unit (DM) used to store long-term information (files).

As the SR memory is built from an ordinary RAM that is part of the machine address space, it is vulnerable to any erroneous program. However, in Enchère, following the strategy advanced in [13], hardware and software mechanisms were designed to make the SR-memory prototype unlikely to be damaged in the event of uncontrolled accesses.

Structure of the SR-Memory Prototype. The SR memory is represented in Figure 8, where

bi ($i \in [1, 8]$) are memory banks, and each *bi* contains 8K bytes;
ri ($ri \in [1, 8]$) are two-bit registers that are associated with *bi*'s and contain current access rights to the concerned memory bank;
AT is an access table made out of 32 bytes—the *j*th bit of the *i*th byte of AT is set to 1 if the *j* memory bank may be addressed (see explanation below).

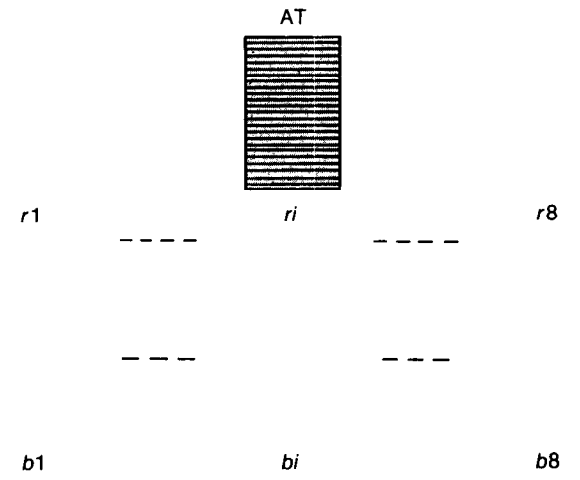

FIGURE 8. SR Memory

Using SR Memory. A process *p* wanting to use SR would first request allocation of a memory bank. This executes the primitive allocate, which

allocates a free bank *bi* ($0 < i < 9$);
searches for a free entry *k* in AT, and sets the *i*th bit of AT[*k*] to 1 and the other bits of AT[*k*] to 0;
returns the physical address (adr) of *bi* and the value *k*—the value *k* will be used later as a key for accessing memory bank *bi*.

Imagine now that *p* wants to write an object located at the address adr0 into an allocated memory bank at the address (adr, *x*) with key *k*. It first executes the primitive *open*(*k*, *w*) where *w* is the write access right. The effect is the following: Register *ri* corresponding to the memory bank *bi* referred to in AT[*k*] is set to the write access right. Then *p* executes the primitive *write*(*adr*0, (*adr*, *x*)), which checks that the memory bank of address adr is opened, and if so, the write operation is executed. After execution of the write primitive, access rights corresponding to the considered memory bank are set to "no right": Actually, the execution of the two instructions *open; write* is indivisible. Although we have used the example of a write operation, the same applies to a read operation.

This very simple hardware mechanism ensures that no direct access to stable memory can be performed; and all access is controlled by means of a key mechanism: Using an erroneous key leads to a memory-access failure.

This mechanism works satisfactorily if every memory bank is accessed by a unique process; however, in our system several processes may share the same memory bank. To ensure that a process does not damage information belonging to another process sharing the same memory bank, the following mechanisms have to be enforced.

The Problem of Information Sharing. In the situation where *n* processes *pi*, ($i \in [1, n]$) sharing the same memory bank, we would like to implement a mechanism such that, if *pi* wants to access *pj*'s ($j \neq i$) stable objects, a memory-access failure occurs.

In the solution we propose, every *pi* possesses a table that provides access to its own stable memory. Protection of this table is ensured using a seal mechanism enforced by cryptography [6]. Entries in the table are encrypted (at the creation of process *pi*) with a key *ki*. Only entries decrypted with the appropriate key deliver significant addresses, and all attempts to decrypt with a bad key lead to memory-access failure.

The above *pi* table (with encrypted entries) is stored in the working space of process *pi*. A nonencrypted version is kept in stable storage and may be used (after encryption) in the event of rollback subsequent to a failure. The Enchère system does not prevent a process from forging an encryption key: Our sole concern is secure table data access.

Stable Storage Management. SR-memory banks are coupled in such a way that every object in the SR memory is represented by two copies. With two coupled memory banks $b1$ and $b2$, only one bank ($b1$ or $b2$) may be granted write access at any time. The system also ensures that the transition (write access for $b1$, read access for $b2$) → (read access for $b1$, write access for $b2$) appears indivisible.

To atomically write an object O from volatile memory to SR (Figure 9), a copy of O has to be made atomically on both memory banks. To do this, O is first copied onto $b1$ (giving $Ob1$), and $Ob1$ is then copied onto $b2$ ($Ob2$). This ensures that $Ob1$ and $Ob2$ are the same. On the other hand, copying O first onto $b1$ and then onto $b2$ could produce unexpected results as O is in volatile memory and might be damaged between the writing on $b1$ and then on $b2$.

Should a crash occur as $b1$ is in *read access right* and $b2$ is in *write access right*, $Ob1$ is not written on $b2$. However, since the system's error-recovery mechanisms ensure that this copy is made at a later time, the write is considered "done." On the other hand, should a crash occur while $b1$ is in write access right and $b2$ in read access right, the copying of $Ob1$ to $b2$ is not possible, and the write operation is considered "not done."

To achieve atomic transfers from SR to disks, we employ a strategy similar to that proposed by Lampson and Sturgis [9], except that, in Enchère, the use of SR memory ensures that objects to be copied onto disks cannot be damaged between transfers as SR is protected against uncontrolled accesses.

The hardware devices used to implement stable storage were designed and developed locally. Performance evaluations of these facilities show that access time to SR memory is twice that to the usual RAM memory, although it is much faster than access time obtained for stable storage built from disks. For example, writing 1024 bytes on our SR memory takes 20 ms (with the I8086 processor), and the same operation in the DFS system takes 140 ms (with the ALTO processor) [11]. Power-failure damages to stable memory are minimized by battery backup to the memory boards.

Commit Protocols and Recovery Algorithms

To ensure coherent update of, and access to, objects in an unreliable environment, we use a model similar to Reed's [15]. In this model, an object is viewed as the history of all the states it has assumed since its inception. Each state is called a version. A version is qualified by its value and a time attribute that specifies the time interval during which the object was in the state represented by this version. The ordering between events is realized by a pseudotime (sequence of integers). A read operation with time attribute t selects the version that has the highest start time lower than t. If the end time of this version is lower than t, it is set to t. A write operation with time attribute t creates a token that has to be committed before becoming a version, if t is greater than the end time of the last version; otherwise it is aborted. The start time of this token is t.

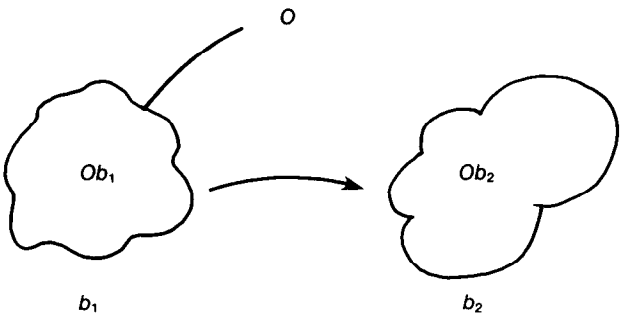

FIGURE 9. Writing an Object in Stable Storage Memory

This simple model provides the basis for synchronization and recovery in decentralized systems. To achieve better performance in Enchère, we simplified the basic model by limiting the number of versions allowed for a given object and allowing a token to be modified by the creating activity.

A particular property of the application that makes it possible to activate a transaction from within another one led us to the implementation of nested activities. This implementation has been fully realized and constitutes an original piece of work covered in detail in [2].

DISCUSSION

Enchère represents the culmination of a 12-person-year effort that resulted in the implementation of a completely distributed operating system.

Although the particularities of this application have somewhat limited the generality of the operating system, which can be qualified as "object oriented," it is possible to devise a system that will handle reliably, not only files, but all types of objects used by processes. This makes us feel that the work presented here could be generalized to produce a general-purpose object-oriented distributed operating system, which is the orientation of our present research.

There are related projects (e.g., EDEN [3], LOCUS [12]) involved in defining and implementing concepts like atomic actions and decentralized control. Our main originality lies in having addressed these research issues in terms of a specific application, which has allowed us to study and experiment with some major operating-system problems, resolution of which proved interesting in their own right.

Acknowledgments. The authors gratefully acknowledge the contributions of B. Decouty, Y. Prunault, and L. Ungaro ("Atelier Micro") and P. Heringer (SOFREL) to the design and implementation of the Enchère system. Thanks are also due to J. le Palmec for his advice during the five years of the project.

APPENDIX

```
type Lot is
   record
      -- suitable fields for lot description
   end record;

type Seller;
type NameOfSeller is access Seller;
type Buyer;
type NameOfBuyer is access Buyer;

task type Buyer is
   entry StartBuyer(Me:NameOfBuyer);
   entry Proposition(Name:NameOfSeller;TheLot:Lot);
   entry Decision(AcceptedOffer:Boolean);
end Buyer;
MaximumNumberOfBuyers: constant Positive := -- suitable value
subtype       PossibleBuyers       is      Natural        range
                                        0..MaximumNumberOfBuyers;
type SetOfBuyers is array(PossibleBuyers)of NameOfBuyer;

task Seller is
   entry StartSeller(Me             : NameOfSeller;
                     NumberOfBuyers: PossibleBuyers;
                     MyBuyers       : SetOfBuyers;
                     Mylot          : Lot);
   entry Offer(Name:NameOfBuyer;Offer:Price);
end Seller;

task body Buyer is
   Me     : NameOfBuyer;
   TheLot : Lot;
   MyOffer: Price;
   Name   : NameOfSeller;
begin
   accept StartBuyer(Me:NameOfBuyer) do
      Buyer.Me := Me;
   end StartBuyer;
   accept Proposition(Name:NameOfSeller;TheLot:Lot) do
      Buyer.TheLot := TheLot;
      Buyer.Name   := Name;
   end Proposition;
      -- display TheLot description
      -- ask the price the user wants to offer
      MyOffer := -- value given by the user
      Name.Offer(Me,MyOffer);
      accept Decision(AcceptedOffer:Boolean) do
         -- display the Decision to the user
      end Decision;
   end Buyer;

   task body Seller is
      Me              : NameOfSeller;
      NumberOfBuyers : PossibleBuyers;
      MyBuyers        : SetOfBuyers;
      MyLot           : Lot;
      Name            : NameOfBuyer;
      Offer           : Price;
      BestOffer       : Price := 0;
      Taker           : NameOfBuyer := null;
```

```
begin
   accept StartSeller(Me              : NameOfSeller;
                      NumberOfBuyers: PossibleBuyers;
                      MyBuyers       : SetOfBuyers;
                      Mylot          : Lot) do
      Seller.NumberOfBuyers := NumberOfBuyers;
      Seller.MyBuyers       := MyBuyers;
      Seller.MyLot          := Lot;
   end StartSeller;
   for I in 1..NumberOfBuyers loop
      MyBuyers(I).Proposition(Mylot);
   end loop;
   for I in 1..NumberOfBuyers loop
      accept Offer(Name:NameOfBuyer; Offer:Price) do
          Seller.Name  := Name;
          Seller.Offer := Offer;
      end Offer;
      if Offer > BestOffer then
          If Taker /= null then
             Taker.Decision(False);
          end if;
          BestOffer := Offer;
          Taker     := Name;
      else
          Name.Decision(False);
      end if;
   end loop;
   Taker.Decision(True);
end Seller;
```


REFERENCES

1. Banatre, J.P., and Banatre, M. Language features for the description of cooperating processes. In *Proceedings of the 4th International Conference on Software Engineering* (Munich, Germany, Sept. 17–19). 1979, pp. 308–314.
2. Banatre, M. Le système Enchère: Une expérience dans la conception et la réalisation d'un système réparti. Doctorat d'Etat, Univ. of Rennes, France, Mar. 1984.
3. Black, A.P. An asymmetric stream communication system. In *Proceedings of the 9th Symposium on Operating Systems Principles* (Bretton Wood, N.H., Oct. 10–13). ACM, New York, 1983, pp. 4–10.
4. Clark, D.D., and Svobodova, L. Design of distributed systems supporting local autonomy. *Dig. Pap. COMPCON* (Spring 1980), 438–444.
5. Ellis, C.S., Feldman, J.E., and Heliotis, J.E. Language constructs and support systems for distributed computing. TR-102, Univ. of Rochester, New York, May 1982.
6. Gifford, D.K. Information storage in a decentralized computer system. CSL-81-8, Xerox Palo Alto Research Center, Calif., Mar. 1982.
7. Gray, J.N. *Notes on Database Operating Systems.* LNCS 60, Springer-Verlag, New York, 1978, pp. 393–481.
8. Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July 1978), 558–565.
9. Lampson, B.W., and Sturgis, H. Crash recovery in a distributed data storage system. Working Pap., Xerox Palo Alto Research Center, Calif., Nov. 1976.
10. Lane, S., Ed. *Proceedings of Electronic Trading of Agricultural Commodities Seminar.* Winnipeg, Canada, Nov. 1981.
11. Mitchell, J.G., and Dion, J. A comparison of two network-based file servers. *Commun. ACM 25*, 4 (Apr. 1982), 233–245.
12. Mueller, E.T., Moore, J.D., and Popek, G.J. A nested transaction mechanism for LOCUS. In *Proceedings of the 9th Symposium on Operating Systems Principles* (Bretton Wood, N.H., Oct. 10–13). ACM, New York, 1983, pp. 71–89.
13. Needham, R.M., Herbert, A.J., and Mitchell, J.G. How to connect stable memory to a computer. *Oper. Syst. Rev. 17*, 1 (Jan. 1983), 16.
14. Organick, E.I. *The Multics System: An Examination of its Structure.* MIT Press, Cambridge, Mass., 1972.
15. Reed, D.P. Implementing atomic actions on decentralized data. *ACM Trans. Comput. Syst. 1*, 1 (Feb. 1983), 3–23.
16. Sporleder, T.L., Ed. *Proceedings of the National Symposium on Electronic Marketing of Agricultural Commodities.* Texas A&M Univ., College Station, Tex., Mar. 1980.

CR Categories and Subject Descriptors: C.2.4 **[Computer-Communication Networks]**: Distributed Systems—*distributed applications*; H.4.3 **[Information Systems Applications]**: Communications Applications

General Terms: Design, Reliability

Additional Key Words and Phrases: auction bidding system, synchronization

Received 9/84; accepted 1/85; revised 3/85

Authors' Present Address: Jean-Pierre Banâtre, Michel Banâtre and Florimond Ployette, IRISA-INRIA, Campus Universitaire de Beaulieu, Avenue du Général Leclerc 35042, Rennes Cédex, France; Guy Lapalme, Université de Montréal, Département de I.R.O., C.P. 6128, Succ. A, Montréal, P.Q., H3C 3J7, Canada.