The design and building of ENCHERE,a distributed electronic marketing system

Jean-Pierre Banâtre, Michel Banâtre, Guy Lapalme, Florimond Ployette

INRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports de Recherche

N° 343

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

Jean-Pierre BANATRE
Michel BANATRE
Guy LAPALME
Florimond PLOYETTE

Octobre 1984

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 - RENNES CÉDEX
FRANCE
Tél. : (99) 36.20.00
Télex : UNIRISA 95 0473 F

PUBLICATION INTERNE N° 237
Septembre 1984
38 pages

THE DESIGN AND BUILDING OF ENCHERE, A DISTRIBUTED ELECTRONIC MARKETING SYSTEM.

Jean-Pierre BANATRE
IRISA-INSA and INRIA, Rennes, France

Michel BANATRE
IRISA-INRIA, Rennes, France.

Guy LAPALME
Département de IRO,
Université de Montréal, Montréal, Canada.

Florimond PLOYETTE
IRISA-INRIA, Rennes, France.

ABSTRACT:

This paper reports an experience in the design and construction of a decentralized system supporting an electronic marketing system. We relate the main steps involved in the design of a computing system satisfying the constraints of the application and the needs of the users. In particular, we describe the properties of the application, its logical structure, the chosen physical architecture and the implementation of a prototype.

We insist on topics relevant to distributed operating systems and put emphasis on research contributions of the project: distributed synchronization, implementation of atomic activities, commit protocols and recovery algorithms.

RESUME:

Cet article décrit une expérience dans la définition et la réalisation d'un système de ventes aux enchères décentralisé. Nous y présentons la démarche employée pour aboutir à un système informatique qui réponde aux contraintes de l'application et aux besoins des utilisateurs potentiels.

Les aspects relatifs à la construction de systèmes opératoires répartis sont mis en avance: nous détaillons les algorithmes de synchronisation, la mise en oeuvre des activités atomiques, les protocoles de reprise.

This work has been partially supported by ADI under contract n0: 82440.

1. INTRODUCTION.

Recently, there has been a growing interest in electronic marketing systems especially for agricultural products [10], [16]. In [16], B. Lindeman Schlei points out that electronic marketing systems provide:

> "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."

To achieve these objectives, an electronic marketing system must be able to handle reliably a high volume of sales at a fast pace through communication links between sellers and buyers distributed over a large area.

In the following, we describe the history and the development of ENCHERE (meaning "auction" in French) which satisfies those criteria of speed, reliability and fairness between buyers and sellers required by the market rules.

To satisfy those needs, ENCHERE implementation combines many original features that were up to now mainly separate research

products: it is a distributed system consisting of a loose network of autonomous microcomputer based workstations communicating with each other via messages. Its reliability is guaranteed by the implementation of atomic actions supported by both new hardware and software products: new stable memory boards were developed to achieve the reliability needs of transactions implemented by dynamic process creations.

This paper describes the evolution of this 5 year applied project where "hard" decisions had to be made concerning the task to be tackled: in 1979-1980 we studied the inner workings of a centralized auction bidding system, and extracted the fundamental principles and rules of the operations. We then defined the software and hardware structure (in that order) of the system, leading to the solution of original problems inherent to this application but that later proved to be interesting for their own sake.

We think it is worthwhile reporting both the history and the building of this project because this should give to a potential distributed system designer some insight into the newly available tools for building such a system and their actual use.

We first describe the problem as we perceived it a few years ago and the requirements the new system would have to satisfy; we then report on the software and hardware structure and their implementation in two steps: first a working prototype to be run on a big "system" (MULTICS) to test the validity of the approach;

a second pre-industrial system was built from the first one, but the main software ideas turned out to be general enough so that they didn't have to be modified.

2. THE ELECTRONIC MARKETING PROBLEM AND OUR "SOLUTION".

2.1 The scenario of electronic sales.

In Europe, there is a long tradition of auction sales for food products through a system called the "dutch clock" which can be roughly described as follows: all buyers are in one room where lots of products are presented for bidding by a sales manager representing the sellers; in front of this room there is a big clock with numbers around it representing possible prices for the lots (for example, they could be numbers going from $2.00 to $0.50 in decrement of $0.05); the sales manager positions the arm of the clock at the starting price of the lot and the arm starts to go down regularly until one of the buyers in the room stops it by pushing a button in front of him; an electronic system identifies who first pushed his button and reports this buyer to the sales manager who can now decide if he accepts this offer or not. This procedure allows a very fast pace: the sale of a lot takes about 10 seconds. This system is well accepted by the sellers and buyers but it requires that all participants be centralized in one place.

In Brittany (France) three such rooms are already linked so that

buyers in one room can bid on the lots presented at each of the three rooms. This helps minimize the rising transportation costs and lessens variations between nearby sale rooms. This scheme permits buyers to have a better choice and sellers to have a greater number of prospective buyers to bid on their lots and, in this way, to expand their markets. In such systems, the order of presentation of the lots is very important because prices vary considerably depending on the time elapsed from the start of the sales: for a variety of reasons, prices usually start low, go up in the middle of the sales and go down at the end. So to be "fair", the system permutes each day the lots of each seller; this causes no problem in the actual system because all sellers are members of the same organization which is responsible for setting up the auction and also compensating sellers whose lots are presented first.

But now there is a need 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, but this will lengthen total sale time. This period is already long (three to four hours) and people participating are business persons whose time is precious and costly. So the question is: how to conciliate 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

usually tends to fill his needs with only few sellers (and often only one). He wants to buy elsewhere only if he cannot find what he needs "locally". So, we propose to create logical sales rooms by giving to each participant (buyer and seller) the necessary tools to deal with only the people he wishes: each buyer will use a "workstation" consisting of a microcomputer linked to the workstation of the sellers through a combination of local and national networks; each station will implement algorithms to achieve communications between the participants which want to deal with one another. The global system, 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.

With this approach, a buyer or a seller uses a workstation consisting of a microprocessor based control unit with a small display, a "dutch clock" and keys to allow him to participate to the sales. There is also an ordinary screen display showing the results of all the transactions being processed in the system. The hardware for each user is the same but the software and the logical meaning of each key is different depending on whether he is a seller or a buyer. Its physical description has been given elsewhere [2], we will only give here its operational aspects, first on the buyer's side.

The buyer can choose the sellers he wants to deal with and can change this choice at any time. The proposition from a seller

is displayed on his control unit and the clock showing the starting price for this lot given by the seller; the clock starts to go down regularly and when the buyer decides that this price is good for him, he pushes a key to send his offer to the seller. He then waits for an answer from the seller telling him if he get the lot or not. He can then receive a new offer, possibly for another seller (which is chosen by the system from the set of sellers this buyer is dealing with).

Of course, the seller sees things quite differently: he pushes a key to send his next proposition to interested buyers and waits for their offer. When all buyers have answered, the value of the best offer is displayed on his control unit; he decides either to accept or to reject this offer and transmits this decision to the buyer. He may now send his next proposition.

2.2 Requirements of a distributed electronic sale system.

The requirements for such a system can be classified in two major categories: those of the application and those of the users.

2.2.1 Requirements of the application

The electronic marketing context imposes by its nature severe constraints on its computer implementation.

1-Sales order: sales activated by a seller are carried out sequentially.

2-Fairness which can be loosely described as follows:

- buyers have an "equal chance" to buy lots proposed by the sellers,
- sellers have an "equal chance" to have their proposition seen at a given time by different sellers, no priority being defined a priori between sellers.

This property is different from the fairness one usually referred to in the distributed systems litterature because it must take into account the competition between the sellers and not only the absence of "starvation". It is a crucial one in our context of commercial marketing and must be enforced in all cases (normal or defective functionning). However, this has to be done while keeping the efficiency of the real-time system which can be measured in terms of availability and speed of processing. It is the direct consequence of our competitive market context in which all sellers have equal rights, i.e. their lots should be equitably interlaced and must not be indefinitely delayed; also important is the fact that the buyers must make their decisions based on the same available information.

3-Reliability: at each session, ENCHERE must provide a complete service throughout the session (several hours a day). Should a failure occur, the buyers and sellers must be able to continue their participation in the sales; of course transaction time may then be lengthened.

4-Extensibility: the architecture of such a system should not be frozen once and for all and should allow the easy insertion and removal of buyers and sellers. This should be possible without any change in the overall architecture.

5-Speed: present systems are known to be too slow because of lack of computing power.

6-Distribution: an auction system may involve people spread over a large geographical area (a region or even a country, possibly more).

2.2.2 Requirements of the users.

An electronic marketing system is intended to be used by different people cooperating (competing in fact) to negotiate their goods. Buyers and sellers have different goals in mind when they take part in such a sale, but at least they share the following requirements which are to be respected by any implementation.

1-Ease of use: the buyers and sellers are not computer specialists and the commands necessary for operating the system should be kept as simple as possible and yet powerful enough to conclude a deal in a short time span (10 seconds).

2-Confidentiality: the system is used in a commercial environment and deals with confidential data relating to the day to day operations of businesses; it must ensures that each

user (buyer and seller) is protected against the others. A buyer must not know the bids of the other buyers except for the price of highest one which is broadcasted to everybody; a seller will receive only the bids relating to his sales and cannot see the bids given by buyers dealing with other sellers. The system must also insure that a user cannot block others or take the place of another one.

3-Autonomy: Every user should feel independent from others: it should be able to join or leave the service according to his own needs.

4-Availability of personal computing power: The duration of auctions is limited (2 or 3 hours per day). The computing power available for the system should be usable for other purposes during the other hours of the day.

All these requirements are the direct consequences of the accepted market rules raised by the competition between users of a "real-world" electronic bidding system.

3. ARCHITECTURE DESIGN.

We now present an architecture for ENCHERE respecting the autonomy and independence of each user while giving an acceptable performance (i.e. execution speed, reliability,...). Two possibilities have been studied:

i) a multiprocessor architecture on a central site which can be

accessed via specialized terminals

ii) a distributed architecture where each user has a personal computer. ENCHERE is then defined as the set of those computers communicating at one moment in order to deal with the sales.

We have to remember that one of the main requirements for our application was the autonomy of the users and flexibility in its use: a user can access or leave the system when he wants to, each buyer can choose the sellers he wants to deal with.

This is very difficult to achieve in a centralized system because in that case, all users are dependent on the site on which the system is implemented. This situation creates unwanted dependencies between users (competitions) which may be unacceptable in business terms. For example, a seller might depend on a buyer or vice-versa even if one is not interested in dealing with the other. This can lead to political problems concerning the access to the system. To solve those problems, we defined an ENCHERE session as a set of autonomous units [4] linked by a communication network. The implementation is not on one site but on as many sites as there are users at a time.

This choice helps to achieve the following objectives:

- the good or bad working of a site only influences the users depending on it for their transactions,
- users can work separately: many sale sessions can go on in parallel as long as their set of users are disjoint.

3.1 Logical structure of the application

The implementation of each sale is modelled around the notion of "activity" [1] [5] composed of a set of cooperating processes (one on each site). So the overall structure of ENCHERE at any time is a "tree" of activities.

There is an initial activity that involves an initial process on each user site. That process is created only when a user wants to participate in the sales. It is responsible for making that user known to the others, for establishing communication links and for creating other processes that will deal explicitly with the sale. These user site processes can join or leave this activity dynamically (i.e. when a user decides that he is not interested in the sales, he can leave; the others continue without him). This activity dies when all users have left. For each lot to be sold, dependent activities are created which involve an instance of a seller process and instances of each buyer processes interested in the sale.

Example:

Consider the following configuration:

-3 sellers sites, SS1, SS2, SS3.

-6 buyers sites, BS1 to BS6.

Dependencies between buyers and sellers are defined by the sets SB(...) of buyers interested in a given seller. Assume that we have:

SB(SS1)={BS1,BS2,BS3},

SB(SS2)={BS4,BS5},

SB(SS3)={BS1,BS5,BS6}

So, for instance, buyers interested in making transactions with SS2 are BS4 and BS5.

Imagine, now, that all three sellers have started a sale, then we have the following activity structure.

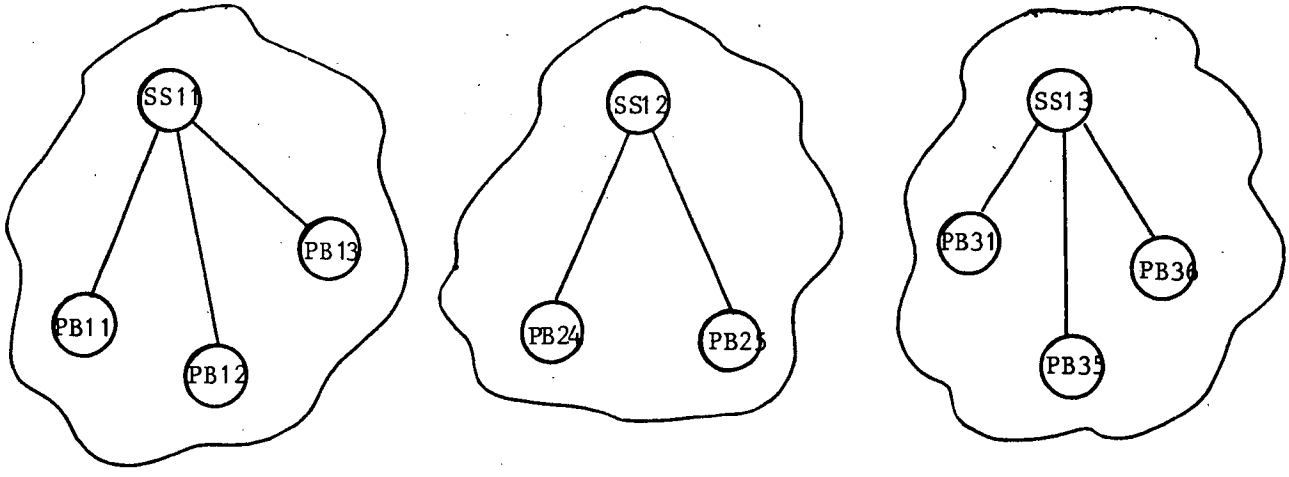

Figure 3.1.

Where:

PBij is the ith instance of a process representing the behavior of buyer BSj. This process belong to the ith activity.

PSij is the ith instance of a process representing the behavior of seller SSj.

So, on each buyer site, there is one buyer process involved in one activity for the sale of a lot from SS1 while another deals with SS3. This activity scheme is very interesting because

all the dynamic properties of the application (i.e. independence, asynchronism, competition for CPU time qr memory management of the processes) are being taken care of by the operating system and are of no concern of the application programmer. Once this tool is available, the sale scenario is very simple as can be seen in the following program written in ADA.

```
procedure MARKET is

   package THE_LOTS is
      type PRICE is delta 0.01 range 0..100.00;
      MAX_LOTS : constant INTEGER := 100;
      type LOT is
         record
             -- suitable fields for lot description
         end record;
   end THE_LOTS;

   use THE_LOTS;

   type BUYER;
   type NAME_OF_BUYER is access BUYER;
   task type BUYER is
      entry PROPOSITION(L:LOT);
      entry DECISION(ACCEPTED_OFFER:BOOLEAN);
   end BUYER;


   task SELLER is
      entry OFFER (NAME:NAME_OF_BUYER; O:PRICE);
   end SELLER;

   task body BUYER is
      ME             :NAME_OF_BUYER; --self pointer
      L              :LOT;
      NUMBER_OF_LOTS:0..MAX_LOTS := 0;
      MY_LOTS        :array (1..MAX_LOTS) of LOT;
      MY_OFFER       :PRICE;
   begin
      accept PROPOSITION(L:LOT) do
         MY_LOTS(NUMBER_OF_LOTS+1):=L;
         -- display the lot description
         -- ask the price the user wants to pay for it
         MY_OFFER := ... ; --value given by the user
      end PROPOSITION;
      SELLER.OFFER(MY_OFFER);
      accept DECISION(ACCEPTED_OFFER:BOOLEAN) do
         if ACCEPTED_OFFER then
```

```
            NUMBER_OF_LOTS:=NUMBER_OF_LOTS+1;
         end if;
      end DECISION;
   end BUYER;

   task body SELLER is
      NUMBER_OF_CLIENTS:0..MAX_BUYERS;
      MY_CLIENTS        :array(1..MAX_BUYERS)  of  NAME_OF_BUYER;
      MY_LOT            :LOT;
      BUYER_OFFER;
      BEST_OFFER        :PRICE;
      TAKER             :NAME_OF_BUYER;
   begin
      for I in 1..NUMBER_OF_CLIENTS
      loop
         MY_CLIENTS(I).PROPOSITION(MY_LOT);
      end loop;
      BEST_OFFER:=0;
      TAKER := null;
      for I in 1..NUMBER_OF_CLIENTS
      loop
         accept  OFFER(NAME:NAME_OF_BUYER; BUYER_OFFER:PRICE) do
            if BUYER_OFFER>BEST_OFFER then
               if TAKER /= null then
                  TAKER.DECISION(FALSE);
               end if;
               BEST_OFFER:=BUYER_OFFER;
               TAKER:=NAME;
            else
               NAME.DECISION(FALSE);
            end if;
         end OFFER;
      end loop;
      TAKER.DECISION(TRUE);
   end SELLER;

end MARKET;
```

Those processes die as soon as one sale has been completed. Note that a buyer can be notified of the rejection of his offer as soon as better one is received by the seller. He can then participate in another sale because he is sure that he will not acquire the lot. The synchronisation mechanism for that "optimization" is taken care by the activity mechanism and does

not have to be dealt within the application.

Another important aspect 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. [2] describe in detail the characteristics and consequences of this indivisibility and atomicity akin to the transaction concept found in database systems [7].

3.2 Physical architecture of the application.

Ideally, every user should be provided with his own workstation. ENCHERE is then implemented by a set of workstations interconnected through a communication system, thus giving the following organization

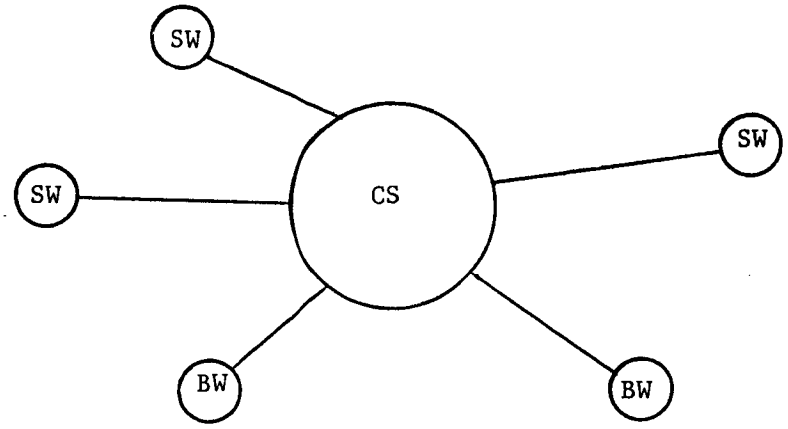

Figure 3.2.

where : SW means seller workstation,
BW means buyer workstation,
CS means communication system.

In the present configuration, every user is provided with a personal terminal made out of two devices: (i) an application

processor (AP) which manages the auctions, and (ii) a workstation (WS) which is the ENCHERE terminal. WS is used for accessing the ENCHERE service. Every WS is connected to an AP. We now give some details about application processors, workstations and communication system.

3.2.1. Application processors.

AP's are built from two micro-processors (I8086 and I8085) connected via Intel Multibus and use disk storage. A stable storage (see section 6) is also connected to the I8086 through Multibus.

The I8086 processor is responsible for the management of auctions and for the management of local peripherals. The I8085 acts as a front processor for the I8086. It manages communication from and to the workstations and the communication system.

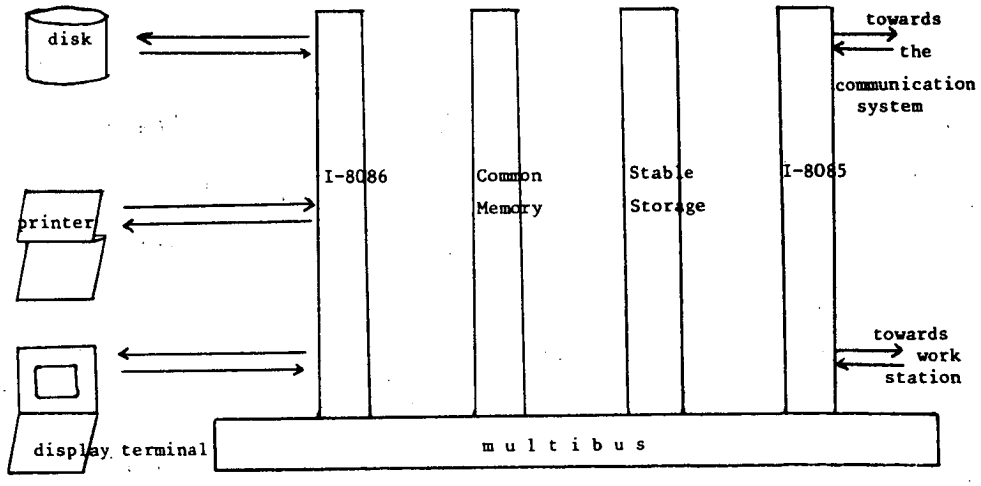

Figure 3.3.

All the hardware choices have been made according to the available hardware at the time of the design. Two storage facilities are used: a common memory and a stable storage which is described in section 6.

3.2.2 The workstation.

The workstation is made of a usual display terminal and of specialized command terminals. A micro-processor manages these devices. The overall organization of the workstation can be visualized as follows:

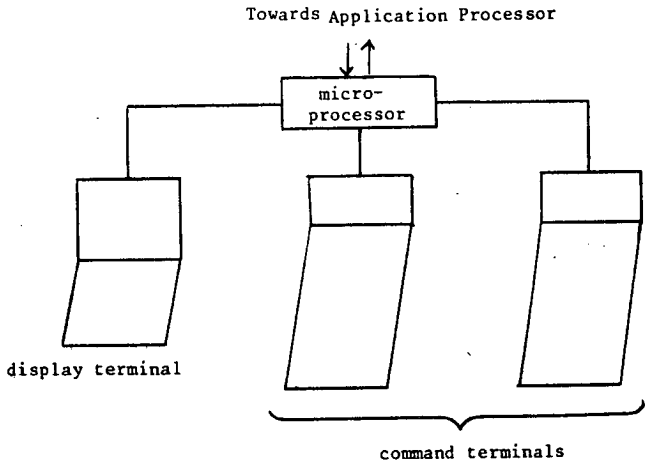

Figure 3.4.

The command terminal provides the user with the necessary commands for participating to sales (new sale, offer, refusal, etc.). Every such command is materialized by a special button that the user pushes in order to signify its desire.

Buyer command terminals are also equiped with a special display which shows instantaneously all information concerning the sale currently processed.

3.2.3. The communication system.

We have chosen not to tie the application to a particular communication system, this because we want to be able to install ENCHERE on any "machine" possessing appropriate features defined by what we call "the minimum interface properties". Among these properties, the communication system should allow message

exchanges with acknowledgment, communication error detection and notification.

4. ABOUT THE CONSTRUCTION OF A PROTOTYPE.

As the proposed system was very new compared with existing ones, it was difficult to foresee the reactions of potential users. However, for various reasons such as credibility, funding of the project, it was necessary to "show" the external characteristics of the new system in order to capture the reactions.

With these considerations in mind, it was decided to build a prototype in two steps:

(1) Construction of a first prototype on a big system (MULTICS in our case) in order to exhibit the external features of ENCHERE. An eventual user, non-specialist in computing, would then be able to "see" how the future system will work. As far as computing was concerned, the system would be centralized.
(2) Depending on the "acceptation" of this first prototype by potential users, a second prototype -a pre-industrial system- could be built.

As the budget for developing these prototypes was strongly limited, some design decisions had to be taken very early in order to make the major part of the software produced for the

first prototype reusable when building the second one. The following decisions were taken:

-the final structure of the system should be "frozen" when building the first prototype. The second prototype should only provide an alternative implementation of the same abstract structure.

- software tools used in the development of both prototypes should be the same. This in order to limit code rewriting.

The two decisions were quite easily implemented. In particular the second one was facilitated by the availability of a local language for micro-processor applications. The compiler of this language produces intermediate code for a virtual machine and depending of the actual target machine appropriate code is generated.

As a consequence of these decisions, the first prototype had to be an emulation of the second prototype.

5. THE IMPLEMENTATION OF THE FIRST PROTOTYPE.

We now report on the emulation of the activity structure on the MULTICS system. After a quick sketch of MULTICS inter-process communication facilities, we describe the main characteristics of the implementation of the first prototype.

MULTICS [14] is a process oriented operating system; when a user logs in, a process is created and associated with his

terminal; this process can create other non-interactive processes called "absentees" which in turn can also create other processes and so on. Those 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 between instances of processes giving the structure illustrated in figure 5.1. We also developed a probe system which allowed us to keep track of all the messages going on at one time. This probe enabled us to simulate break downs and time-outs and evaluate the effects of software or hardware failures on the system.

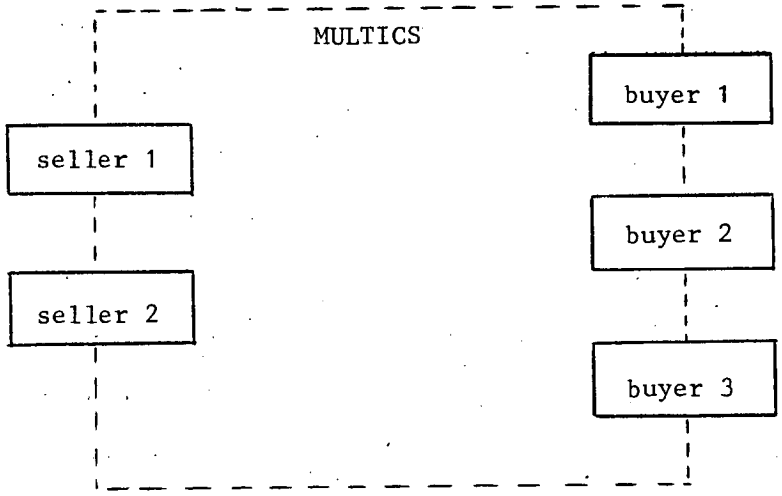

Figure 5.1.

Users are linked via MULTICS. Each logical entity associated with a user incorporates many modules, a few of which are implemented on the microcomputer of the workstation and the others are created under MULTICS. Each of the boxes of Figure

5.1 can be subdivided as shown in Figure 5.2.

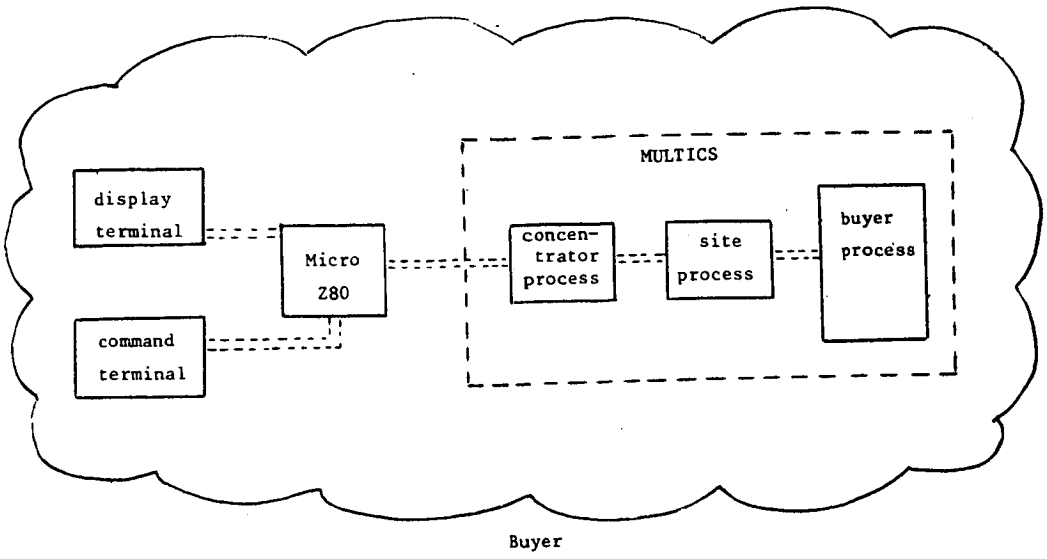

Figure 5.2.

We indicated earlier that the software structure of the system was based on the notion of activity. We now sketch the implementation of this notion under MULTICS. For example, consider the sale of a lot from a seller "S" to two buyers "B1" and "B2". Each site is represented by a "site process" which interprets commands issued by the user. Thus we have three site processes: SS for site S, SB1 and SB2 for sites B1 and B2. When a command "new-sale" is issued on site S, the process SS signifies that a new activity involving himself and SB1 and SB2 has to be created. 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 in order

to form a new activity as summarized on figure 5.3.

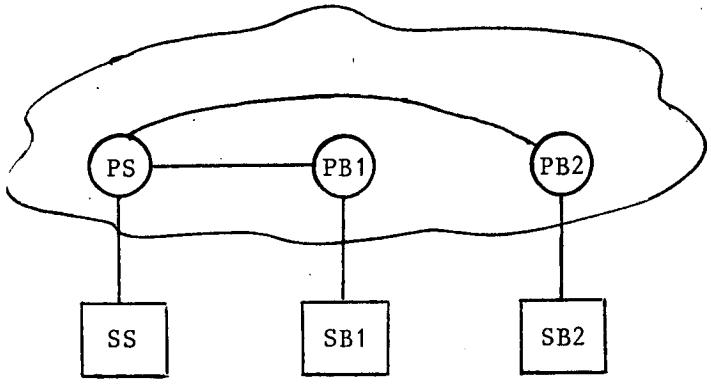

Figure 5.3.

A buyer site may create many instances of processes PBi, one for each sale which can go on in parallel (ie: activated by distinct sellers). This approach is costly, but it is simple and reflects very well the software structure we had decided upon.

This first prototype was very important in the project for two main reasons:

- we were able to quickly have a working model of the system and so we have users' feedback on the system
- we could also validate our structuring methods; this can be explained by the power of the concepts implemented by MULTICS and their good adequacy with our choices.

Unfortunately, this prototype did not permit a fully satisfying quantitative evaluation of the service because of the time

sharing nature of the system in which external users to ENCHERE can greatly influence the response time to the commands.

The main lessons learned from this first experiment were:

- the original idea of having the identical workstation keyboards for both sellers and buyers was not a good one from the ease of use point of view. We had to give too many functions to each participant. So we designed two different keyboards in the second experiment.
- we should always strive for simplicity. Our original goals were too high for the first prototype, we should have been satisfied with only a small demonstration unit giving an external view of the system and not try to put in place all the activity structure which proved to be too expensive to use in real-time. But the test of its logical feasibility was important enough to warrant an experimental implementation but one that should have been independent of the demonstration unit.

This part was well accepted and prompted us (and others who put money into the project) to continue in the same vein to develop the system. It was now feasible to implement both site and sale dealing processes on a network of microcomputer linked together first via MULTICS which then will be confined to a "safe" network role and then via a combination of local and national networks.

6. THE IMPLEMENTATION OF THE SECOND PROTOTYPE.

The second prototype consists of three application processors and five workstations connected, in a first stage, through a concentrator (Intel 8086). As said earlier, ENCHERE is so designed that no problem should arise when transported onto a network.

In this section, we do not detail the construction of the system. Instead, we concentrate on two original contributions which arose while building the system, namely:

i) Implementation of fairness,

ii) Implementation of atomic transactions.

6.1. Implementation of fairness.

Fairness as described in section 2 can be expressed the two following properties:

Property 1: The system must present the propositions to various buyers in the same order for each seller.

This ensures that buyers may elaborate their strategy according to the same informations. On a technical viewpoint, this property is crucial in the avoidance of deadlocks as a buyer must terminate a sale before participating to a new one. Imagine the situation where two lots L1 and L2 are to be sold, L1 proposed by a seller S1 and L2 by S2. Two buyer's B1 and B2 compete for these lots. If B1 bids on L1 first (transaction T1) and B2 bids on L2 first (transaction T2), then neither T1 nor T2 will ever be completed because in order to be completed T1 needs a bid form B2 which cannot provide it before completion of T2

(usual deadlock situation).

Property 2: No priority is given by a buyer to the propositions of a particular seller for a given buyer. This ensures "justice" among participants.

6.1.1. Implementation of property 1.

A timestamp 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 to deal with, the system can always take the one having the smallest timestamp (and in the case of equality, an order can be given on the name of the sellers; Lamport [8] shows a way to achieve this in a "fair" way without necessarily defining a priori a total order on the names).

The requirement saying that a seller must have finished with a sale before starting another one, imposes that these local choices, made by the computers on each buyer site, be coherent. In particular, it is necessary to determine if this buyer has already received the "oldest" proposition before making the choice.

To fulfill these requirements, we have shown in [2] that it is sufficient that each buyer-site B selects at his local time H(B), the proposition which posesses the smallest timestamp lesser than H(B)-DT. (DT is the maximum transmision delay between two sites).

6.1.2. Implementation of property 2.

The solution proposed above is not powerful enough to implement property 2; this can be easily shown with an example:

S1, S2, S3: seller sites

Bi : buyer site linked to S1, S2 ans S3.

S1 starts a sale (s11) with his local time h1=10

S2 starts a sale (s12) with his local time h2=15

S3 starts a sale (s13) with his local time h3=20.

The technique of the preceding section tells Bi to choose s11. Suppose 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 Bi chooses s12. When s12 is finished, Bi must choose between s11´ and s13. The method given earlier forces to take s11´ thereby violating property 2.

To fulfill the requirements of property 2, Bi dealing with sales of many sellers must be able to "compare" the times given by the local clocks of these seller sites: i.e. if at a given time H(Bi), two sales´ timestamps of sellers Sj and Sk are H(Sj) and H(Sk) such that for Bi (H(Sj)<H(Sk)) the next sale started by Sj will be stamped by H(Sj)´>H(Sk). In our example, s11´ should be dated by a time greater than 20.

To solve this problem, we introduce a notion of "fuzzy" time

which is defined as follows: the local clock of user sites U1,...,Un show the same fuzzy time than the local clock of a site Ui if and only if |H(Uj)-H(Ui)|<dt. (dt is the maximal drift between the clocks of user sites Uj and the one of Ui).

If we take for granted that the length of a transaction is greater than 2*dt and that all seller sites for a given buyer have the same fuzzy time, then we prove that property 2 is guaranteed [2].

The situation where these conditions are not fulfilled has been studied with great care. In particular algorithms which allow clock resynchronization following misfunctionning of a site have been proposed.

6.2. Implementation of atomic transactions.

The system is intended to resist to the following types of faults:

-software faults which may lead to undesirable actions resulting for example in uncontrolled memory accesses.

-various hardware faults such as power failure which may also result in uncontrolled memory accesses.

The crucial aspect of uncontrolled memory access has been recognized after the construction of a first prototype of ENCHERE. These observations revealed that many failures led to bad memory accesses and that an effort should be made in order to avoid memory degradation.

This section describes three aspects of the work that has been performed in order to achieve high reliability:

-Design and implementation of stable storage,

-Implemention of commit protocols,

-Recovery algorithm,

6.2.1. Design and Implementation of stable storage.

Stable storage [9] is generally viewed as a memory device with the following properties:

i)-The physical storage is stable (i.e. information does not decay over time),

ii)-The write operation is atomic.

Usually, stable storage is built from disks. Although disks do not provide directly stable storage, they possess properties from which stable storage can be implemented.

In ENCHERE, each application processor is equipped with a stable storage which may contain two types of objects:

i)-Objects which may be accessed directly by processes. Such objects are generally of a small size, they are for example variables. Their lifetime is that of the activity which created them.

ii)-Objects which are stored for a long period for example files.

It is clearly unrealistic to store objects of type (i) on devices such as disks, because the time necessary to access them

would be too long. In order to cope with this problem our stable storage is made out of two devices:

i)-A stable RAM unit (SR) which is a part of the machine address space.

ii)-A disk unit (DM) which is used to store long-term information (files).

As the SR memory is built from an ordinary RAM which is a part of the machine address space, it is vulnerable to any erroneous program. We describe the hardware and software mechanisms that have been designed in order to make our SR memory prototype unlikely to be damaged even in presence of uncontrolled accesses. In that, we follow the same idea as that put forwards in [13].

6.2.1.1. Structure of the SR memory prototype.

The SR memory is represented on figure 6.1., where:

-bi (i$\in$[1,8]) are memory banks. Every bi contains 8K bytes.

-ri (ri$\in$[1,8]) are two-bit registers associated with bi's. They contain the 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, (as explained in the next section).

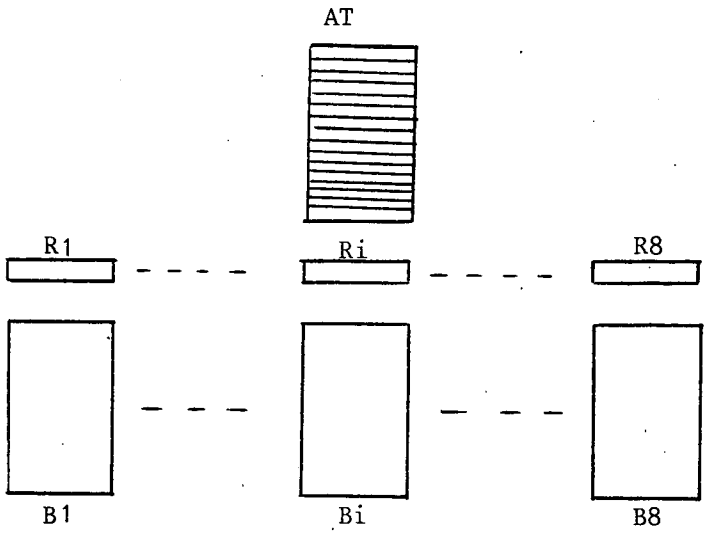

Figure 6.1.

6.2.1.2. Using SR memory.

A process p wanting to use SR should first request the allocation of a memory bank. It 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, the other bits of AT[k] are set to 0,

-returns the physical address (adr) of bi and the value k, k will be used later as a key for accessing the memory bank bi.

Imagine now that p wants to write an object at the address adr0 into a allocated memory bank at the address (adr,x) with key k. First, it executes the primitive open(k,w) where w is the write access right. The effect is the following: the 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(adr0,(adr,x)). This primitive checks that the memory bank of address adr is opened, if so, the write operation is executed.

After execution of the write primitive, the access rights corresponding to the considered memory bank are set up to "no right". Actually, the execution of the couple of instructions (open; write) is indivisible. We have taken the example of a write operation, the same applies to a read operation.

This very simple hardware mechanism ensures that

i)-no direct access to stable memory can be performed,

ii)-any access is strongly controlled by means of key mechanism: using an erroneous key leads to a memory access failure.

This mechanism works satisfactorily if every bank may be accessed by a unique process, however in our system several processes may share the same memory bank. The above mechanisms have to be enforced in order to avoid that a process may damage informations belonging to another process sharing the same memory bank.

6.2.1.3. The problem of information sharing.

Consider n processes pi, (i$\ddot{e}$[1,n]) sharing the same memory bank. We would like to implement a mechanism such that if pi wants to access pj's (j≠i) stable objects, a memory access failure occurs. The solution we propose may be described as follows:

Every pi possesses a table which gives access to its own stable memory. The protection of this table is ensured by using a

seal mechanism enforced by cryptography [6]. Entries in this table are encrypted (at the creation of the process pi) with a key ki. Only entries decrypted with the appropriate key deliver significant addresses, all attempts to decrypt with a bad key lead to memory access failure.

The above table (with encrypted entries) is stored in the working space of the process pi. A non-encrypted version of this table is kept in the stable storage and may be used (after encryption) in case of rollback subsequent to a failure.

Actually our system does not prevent a process from forgering a encryption key. Our sole concern is about secure table data access.

6.2.1.4. Stable storage management.

SR memory banks are coupled in such a way that every object in the SR memory is represented by two copies.

Consider two coupled memory banks b1 and b2. At any time only one bank (b1 or b2) may be granted write access. The situation where b1 and b2 be granted simultaneously write access is forbidden by the system which ensures also that the transition (write access for b1, read access for b2)-->(read access for b1, write access for b2) appears as indivisible.

Let us now examine the problem of writing atomically an object O from volatile memory to SR. Figure 6.2. may represent the situation:

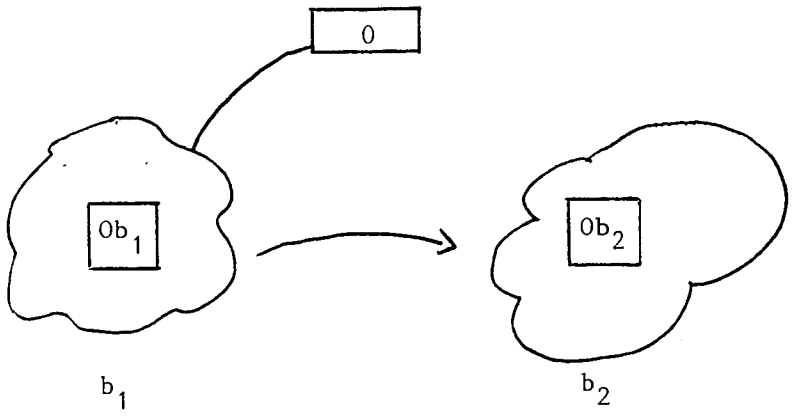

Figure 6.2.

A copy of O has to be made atomically on both banks. we proceed as follows:

O is first copied onto bl (giving Obl) and Obl is copied onto b2 (Ob2). This ensures that, in correct situation Obl and Ob2 are the same. Copying O onto bl, then onto b2 could have given unexpected results as O is on volatile memory and could be damaged between the writing on bl and that on b2.

Imagine now that a crash occurs in the following situations:

i)-bl is in read access right, b2 in write access right. This means that Obl is not written on b2. Error recovery mechanisms implemented by the system ensure later that Obl be copied onto b2. The write is considered as "done".

ii)-bl is in write access right, b2 in read access right. The copy Obl cannot be done, and the write operation is considered as "not done".

Concerning atomic transfers from SR to disks, we employ a strategy similar to that proposed in [9]. However, the use of SR memory ensures that the objects to be copied onto disks cannot be

damaged between transfers as SR is protected against uncontrolled accesses.

Hardware devices necessary for implementation of stable storage have been designed and developed locally. Performances of these facilities have been evaluated. Access time to SR memory is twice higher than access time to usual RAM memory. It is much faster than access time obtained for stable storage built from disks: for example, writing 256 bytes on our SR memory takes 0.5ms (with I8086 processor) and the same operation in the DFS system takes 9ms (with ALTO processor) [11]. Power failure damages to the stable memory are minimised by battery back-up to these memory boards.

6.2.2. Commit protocols and recovery algorithms.

We have concentrated on the design and implementation of protocols which ensure the atomicity property for activities. In particular, we must ensure coherent updates of and coherent access to objects in an unreliable environment. In order to do this, we have used a model somewhat similar to Reed's one [15] and realized a complete implementation.

A particular property of the application, the possibility of activating a transaction from within another one, lead us to the implementation of nested activities. This implementation has been fully realized and constitutes an original piece of work described in [2].

7. DISCUSSION.

This paper summarizes an effort which took 12 men-years and lead to the implementation of a complete distributed operating system. We have tried to describe the main steps involved in the development of the system, from the application to the architecture. We particulary insist on the development of prototypes and exibit some current research issues which have to be tackled in order to produce a realistic implementation.

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

Concerning related work, some projects currently under development are defining and implementing concepts such as: atomic actions, decentralized control, etc... Let us mention for example EDEN [3], LOCUS [12], our main originality, stands in the fact that the research issues we have treated have been geared by a specific application, which allowed us to study and experiment some major topics concerned with operating systems.

ACKNOWLEDGEMENTS:

The authors gracefully acknowledge the contributions of B.DECOUTY, Y.PRUNAULT and L.UNGARO ("Atelier Micro") and Ph.HERINGER (SOFREL), in the design and the implementation of the ENCHERE system.

REFERENCES:

1. BANATRE J.P., BANATRE M.
Language Features for the Description of Cooperating Processes.
Proc. 4th. Int. Conf. on Soft. Eng., Munich, Sept. 79, pp.308-314.

2. BANATRE M.
Le Système ENCHERE: une Expérience dans la Conception et la Réalisation d'un Système Réparti.
Thèse d'Etat, Université de Rennes, Mars 1984.

3. BLACK A.P.
An Asymmetric Stream Communication System
Proc. of 9th Symp. on Operating Systems Principles, Bretton Wood, Oct. 83, pp.4-10.

4. CLARK D.D., SVOBODOVA L.
Design of Distributed Systems Supporting Local Autonomy.
Dig. Papers COMPCON Spring'80, 1980, pp.438-444.

5. ELLIS C.S., FELDMAN J.E., HELIOTIS J.E.
Language Constructs and Support Systems for Distributed Computing.
University of Rochester, TR-102, May 1982.

6. GIFFORD D.K.
Information Storage in a Decentralized Computer System.
CSL-81-8, Xerox Palo Alto Research Center, March 1982.

7. GRAY J.N.
Notes on Database Operating Systems.
LCNS 60, Springer-Verlag, 1978, pp. 393-481.

8. LAMPORT L.
Time, Clock and the Ordering of events in a distributed System.
CACM 21,7, July 1978, pp.558-565.

9. LAMPSON B.W., STURGIS H.
Crash Recovery in a Distributed Data Storage System.
Working Paper, Xerox PARC, Nov. 1976.

10. LANE S. ed.
Proceedings of Electronic Trading of Agricultural Commodities Seminar.
Winnipeg, CANADA, Nov. 1981.

11. MITCHELL J.G., DION J.
A Comparison of two Network-Based File Servers.
CACM 25,4, April 1982, pp. 233-245.

12. MUELLER E.T., MOORE J.D., POPEK G.J.
A Nested Transaction Mechanism for LOCUS.
Proc of 9th Symp on Operating Systems Principles, Bretton-Wood, Oct 1983, pp.71-89.

13. NEEDHAM R.M., HERBERT A.J., MITCHELL J.G.
How to Connect Stable Memory to a Computer.
op. Syst. Rev., Vol 17,1, Jan. 1983, pp.16-16.

14. ORGANICK E.I.
The Multics System: An Examination of its Structure.
M.I.T. Press, 1972.

15. REED D.P.
Implementing Atomic Actions on Decentralized Data.
ACM Trans. Comput. Syst. Vol. 1,1. Feb. 1983, pp. 3-23.

16. SPORLEDER T.L., ed.
Proceeding of the National Symposium on Electronic Marketing of Agricultural Commodities.
Texas A&M University, Dallas, March 1980.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique