Microcomputer-Based Distributed Processing

John J. Tibbetts
Vice President, Research & Development
The DATALEX Company

INTRODUCTION

If one were to attempt to list the major technological changes of the last decade, surely at the top of that list would be the so-called microcomputer revolution. Over the last 10 years intelligent devices have jumped from the research and development labs out into the hands of scientists and engineers and then into business and then into small business and the home. I believe that the depth to which microcomputers have penetrated our society would have surprised even the most adventurous of futurists of 10 years ago. Nor does it appear that this wave of change is at all slowing down. Projections show very rapid growth, both in the home and business, over the next five years.

Also interesting to observe is the evolution of the packaged microcomputer as it exists today. Ten years ago the first microprocessor chips were just coming out. A few years later, a few poorly capitalized companies originated the packaged microcomputer with a microprocessor, support chips, power supply, and a limited amount of memory. Over the next couple of years the microcomputer was really in the hands of the electronics buff and the amateur radio operator with very little emphasis being placed on software. The professional computer person of five years ago approaching a microcomputer was faced with theprospect of working with a virtually bare machine in terms of professional software tools. But in the last five years there has been a rapid growth of professional software, including commercial-grade operating systems, languages, and some applications software packages. Where a few years before the microcomputers were the province of the hardware junkies, now even the major microcomputer manufacturers had very much awakened to the role of software in selling computers.

A striking poster that is now being distributed by Apple internally as well as to its sales outlets, says in massive letters "SOFTWARE SELLS SYSTEMS."

However, despite the large number of software products that are now avialable, there are actually a relatively small number of applications represented. Pick up a BYTE magazine (that is, if you can. What started out a a pamphlet now runs 500 pages, most of which are ads) and perform the following exercise. Take a clean sheet of paper and start writing down software products, grouping them according to function. You will find an interesting pattern. There will be a large number of games. There will be a number of packages which are small business accounting packages. There will be a preponderance of word and text processors, some of which are quite good. There will be some financial modeling software in the Visicalc and Plan 80 sense, used for limited, but very interactive, financial modeling applications.

There will finally be a group of programs calling themselves database management systems. I hesitate to call true database management systems in the classical sense of the word. Some of them are certainly capable file management systems.

What seems to be almost totally missing are products which are important to the kind of people who attend these meetings, that is, people with larger computer needs. What is missing is software that emphasizes the nonpersonal use of personal computers. Let's examine what a few of these software product categories might be:

1. *Communications software.*
   Communications software accurately moves transactions or files of data back and forth from your microcomputer to your corporate computer. It is true that there are some communications packages which do exist for microcomputers, but I would assert that very few of them are oriented toward commercial grade data handling chores which should include such features as error detection and retry, bidirectional control of the communications stream from either the microcomputer or the host computer, and the ability to accommodate full binary transfers of data. These matters are discussed in much greater detail in another talk which I am giving at this meeting.
1. *Intelligent terminal software.*
   By intelligent terminal software, I mean software which can format and edit transactions of data as they are being entered and before they are submitted to your corporate data processing machine or network. Nearly all online, realtime applications which are performed onto an HP3000 are performed via dumb terminals. Consequently, all of the editing and transaction formatting needs to be done by the central computer. Not only can this be slow from a processing point of view, it can also be very slow from an apparent operator speed point of view. The use of intelligent terminals software

running on industry standard microcomputers would allow increased capability and a higher performance in interactive applications.

3. *Data entry software.*
   By data entry software I mean software similar to the intelligent terminal software just described but which can act independently of the remote computer entirely. That is, the transactions as they are gathered are stored locally — usually on a diskette on a microcomputer — and are maintained on the microcomputer until such time that all of the data has been entered. This type of software solves the standard data entry needs of data-intensive commercial applications.

In this talk, I will address myself to microcomputer software being used as intelligent terminal and data entry software and relate to you some of my experiences in this regard.

BUILDING BLOCKS

Before we can discuss the particulars of intelligent terminal and data entry software, it would be important to first define some of our terminology in terms of industry-standard microcomputers and industry-standard software for microcomputers. In this section, we will define some of the assumptions we have used in the building of our software systems.

Let us first consider hardware. Microcomputers can of course come in many sizes and shapes, all the way from the little $200 Sinclair Microcomputer on the low side to the very expensive microcomputers bordering on minicomputers on the high side. In general, I feel that the standard minimum configuration for a general purpose data entry based microcomputer to be a 64K system or greater, with floppy disk support. The reason I recommend a full 64K system for your microcomputer is simply that the cost of main memory has dropped to the point where the software costs associated with working in smaller memory sizes outweigh the amount you spend on memory unless you are producing a special-purpose, high volume data entry product. It is interesting to note that many of the microcomputer manufacturers have moved from a position of giving you whatever memory size you would like, to recommending and then strongly recommending 64K systems, and now some of them are selling only that configuratin. Although Winchester disk technology is a very exciting element in today's microcomputing, for data entry and intelligent terminal based applications it usually isn't necessary except in two specific cases: one in which you have multiple systems, perhaps more than 3 or 4 in the same location, which you would like to share the common systems software from a single Winchester drive; or two, in cases in which there are going to be keyed lookups into larger data structures which would required the performance you can get from a Winchester drive rather than a floppy disk drive. Printers are sometimes useful in certain distributed applications but in most cases don't seem to be required.

You will notice that I make no specific recommendations on hardware manufacturers. This is because I have come to the conclusion that the main reality about microcomputer hardware is that it is in a very dynamic state. What you want in microcomputer hardware for distributed processing applications is hardware that is *maintainable, reliable, and, probably, from a large-name vendor*. Beyond that no specific recommendations on my part are advisable. The fact that I am preparing this talk two months in advance of giving it leaves plenty of room for more significant announcements to be made before the talk is even presented.

Perhaps even more important than the selection of hardware for a distributed processing application is the selection of your software operating system. I say this for two reasons:

1. If the operating system is portable enough, it will alow you to change your decision about hardware during the development of your application or during different stages of its implementation.
2. The operating system has a much greater influence over the programming techniques and systems capabilities than does the hardware in which it is packaged.

Now, what kind of microcomputer operating systems can we expect to find these days? Let's perform a mental exercise. I will entitle this exercise, "Name That Operating System." OK, name this operating system:

- Runs in a Stack Environment
- "Segmented" Architecture
- Non-Von Neumann → Code Segments Separate from Data Segments
- Up to 256 Code Segments of 65K Bytes Apiece
- Process-handling
- "Intrinsic" Procedures to Implement Supervisor Calls
- Inter-linkable Languages

Do you have the name of that operating system fixed in your minds? Good. Now, let me add a few more attributes to the list.

- Runs on 8080, 8085, 8086, 8088, Z80, Z8000, 6800, 68000, 6502, LSI-11, TI-9900
- Has 70,000 licensed users
- Supports PASCAL, FORTRAN-77, BASIC (interlinkable)

Do you still have the same operating system in mind now? The operating system I have been describing has many attributes asociated with the HP3000 MPE architecture and the Burroughs architecture before it. This operating system is the UCSD p-System, so-called because it originally developed from the PASCAL language project from the University of California-San Diego. It is now marketed worldwide by SofTech MIcrosystems. I consider this operating system to be the most professional of the 9-bit/16-bit microcomputer operating systems. It gives the deveoper a capability approaching the power of MPE running in a portable microcomputer environment.

The word "portability" can't be stressed enough when dealing with microcomputing. I meantioned previously that hardware is in an extremely dynamic state in the microcomputer industry. Thus, the notion of protecting your software investment which HP has always preached to their customers is extremely important in the microcomputer domain. After all, in the microcomputer domain your software investment is often many times the cost of the hardware for small hardware configurations and the importance of protecting it against the extremely volatile hardware changes we find in the microcomputer area is very important. UCSD p-System portability means that we can take compiled, running systems and move them from microcomputer to microcomputer, even running different processors, and have them immediately execute. This is true portability.

Another building block we need for distributed processing type applications is good, commercial-grade communications software. We need software that can move transactions or files of data back and forth to the HP3000 with full error detection. Our approach has been to write compatible communications programs, both on the microcomputer and the HP3000, which provide for sending checksummed packages of data and messages back and forth between the two processors. Thee is a great deal more versatility and reliability when you have interlinked programs running on both sides. These programs will run very effectively even over noisy telephone lines or in environments in which characters, such as the important DC1 character, may suddenly disappear. The programs have the ability to time-out after priods of no communications so that the error recovery can be graceful. The programs also have the ability to allow either side — that is, either the local microcomputer or the remote HP3000 — to control the communications. Thus, for instance, we have built applications in which the operator simply starts up an HP3000 UDC, the UDC starts up perhaps a COBOL transaction processor which polls the microcomputers for the filenames which they need to send and then requests that the files be sent.

The last building block needed for distributed processing systems is a comprehensive forms language. To date, such software has not been available on microcomputers and consequently we have spent the last couple of years building it ourselves. The forms language tends to have many of the attributes of the V/3000 approach of forms building; to wit, draw a pickture of the form in a screen editor and then proceed to specify attributes, such as range checks, table lookups, optional fields and so forth about the form. The significant difference between a forms language that can be written on a dedicated microcomputer and one that can be written on a larger shared processor is in the greater degree of user interaction that can be accomplished on a microcomputer. On a keystroke-by-keystroke basis, the microcomputer can do instantaneous editing of the data, rather than waiting to gather up a whole block of data and then transmit it to some computer somewhere else for editing. This means that the microcomputer forms have a very high apparent speed, no matter what the speed of the remote processor.

Other form attributes that have been implemented are:

- A "dup" key. This key immediately copies the previously typed entry to the current data value.
- Function keys which can cause immediate action in the data, such as default values or clearing a field.
- Data verification in the IBM sense of retyping the data exactly the same way (just the way your key punchers have been trained to do it).

With these building blocks we have the tools needed to build very innovative and effective data processing nodes onto existing information network.

INTELLIGENT TERMINALS

Our approach to writing intelligent terminal software has been to use our forms language to build, compile, and maintain the forms on a microcomputer. The compiled forms can be stored on the host HP3000 for distribution. These are periodically distributed to the various nodes through the communications software. The applications program, say a COBOL program written on the HP3000, controls the microcomputer by sending down very simply formatted ASCII strings to the microcomputer to give it its instructions. Since there are no special control character sequences, the screen commands can be dispatched by any language, not just COBOL. They can even be dispatched by UDCs. For instance, the ASCII string ".CS" tells the remote microcomputer to clear the screen. The command ".LF PRODUCT" tells the remote microcomputer to load the form named "PRODUCT" from the floppy disk into memory. Since the forms live locally on the microcomputer, a form change command represents only 10 characters transmitted from the HP3000 to the microcomputer. Compare this with the 1,000 to 2,000 characters that are usually required to change a form on a non-intelligent computer. On a typical floppy disk system we usually can store from tens to hundreds of forms, depending on the capacity of the floppy disk. Once the application has displayed the form on the microcomputer, it can give them a simple command such as ".GF" to get the form. This command causes the microcomputer to issue a read for that form and does all of the local form editing and the microcomputer without any involvement by the host computer. The result is a very high apparent screen speed that is being controlled by the remote computer.

There is a very close parallel between the intelligent terminal command strings and the equivalent subroutine calls that one would issue from a V/3000 system. Thus, one can either read or write whole forms or individual fields or any combination of them. In addition, using intelligent terminal software, one can do some fairly intelligent operations on the screen. For instance, we can request that only modified fields on a form be sent back to the host computer with some identifier on each field. Another intelligent operation is the reformatting of the record on the fly, such that the fields themselves can be shifted in position with various constant data inserted into the transmision stream.

There are a couple of easily definable benefits from using this kind of intelligent terminal software:

1. *Performance.*
   Using an intelligent terminal improves the performance of the program on the HP3000 in that it does not need to be burdened with a lot of editing operations that can be done immediately by the local microcomputer. By the time the data is sent to the HP3000, it is as clean as local editing can provide. This system also performs very well for the operator who gets the benefits of immediate error checking on those fields that have had defined local microcomputer editing.
2. *Portability.*
   This system is very portable both with respect to the program running on the host computer — that is, the HP3000 — and to the program running on the microcomputer. For the host computer, since all of the commands for the screen operations are simple ASCII strings, the COBOL programs tend to be far more portable than COBOL programs with embedded forms control procedures. If one wished to take a COBOL program and move it to an IBM main frame, the only conversion required would be the standard conversion of any COBOL program from an HP3000 to an IBM system. On the microcomputer side, the intelligent terminal software — since it has already been defined to be very portable on microcomputers — can be running on an HP125 or on an APPLE II or any of the other microcomputers which support the p-System operating system. This means that the same application program can drive a variety of microcomputer-based intelligent terminals, depending on the preference of the system implementer or perhaps what hardware might be existing in the office that this system is running into.

One last note on the intelligent terminal software. Even with the benefits I have just described, unless a user has some particular need for portability or higher performance, my best guess is that the advantages would not be sufficient to cause someone to establish an intelligent terminal network instead of using dumb terminals. The real usefulness of this intelligent terminal software will come to light when we begin talking about offline uses of the microcomputer and especially their hybrid usages.

OFFLINE DATA ENTRY

Perhaps the single most significant application of the new microcomputer technology for users of existing information networks is doing offline data capture. This means that we can have our microcomputers sitting either in our data entry departments or in remote offices offline from our HP3000, gathering data, putting it onto floppy disks, doing local editing as previously described, perhaps performing batch balancing, perhaps generating proof listings of the data so that it can be visually verified, or rekey verified by the data entry operators, and then have the batches closed and transmitted for processing to the remote computer.

We see two major users of microcomputer-based data entry. The first is in collecting volume data typically entered by the data entry function of your information system. Let's quickly compare a microcomputer solution to the standard existing solutios for gathering data:

1. Compare to collecting data on cards or with a key-to-diskette system such as the 3741, the microcomputer can do a much more comprehensive job of editing the data. It not only can do checks on the type of the data but also on particular values of the data or by comparing values in several fields of the data and so forth. Furthermore, the microcomputer using a formatted CRT-type technology can much more readily be operated by users than just by the professional data entry operators. A recent survey by a professional data entry association shows that approximately 70% of corporate data processing departments are shifting to user data entry from centralized data entry. In terms of performance, the speed of key entry into a microcomputer usually exceeds, and sometimes by a considerable factor, the entry speed through cards and key-to-diskette systems (due to the fact that fairly smart duping operations can be programmed which can minimize the key strokes that need to be entered).
2. Compared to intelligent key-to-disk systems, the microcomputer would roughly equal them in terms of key entry performance since these machines also are intelligent and programmable and can provide for very smart data collection algorithms. The principal disadvantages of the key-to-disk systems, which are usually characterized by a minicomputer with a cluster of terminals, is that the per-terminal cost of the microcomputer is considerably less than the per-terminal cost of the clustered mini when you are dealing with fewer than 8 or 10 terminals in a specific location. For large scale data entry chores, one would probably still favor a clustered mini for doing the data entry chores. For operations with only a few stations or where the stations are distributed, the mi-

crocomputer again comes out as being a more favorable solution.

3. Compared to online data entry using V/3000 or some other online screen formatted technology, the microcomputer scores much higher in keystroke performance and operator performance as well as not burdening the machine with keystroke intensive work. I think it is the common experience of the HP3000 community that having several data entry operators entering data online disproportionately burdens that system's performance.

Again the issue of portability and versatility needs to be made in a comparison of data entry approaches. Virtually all other data entry gear is single purpose equipment. If you buy a key punch machine, or a 3741, or a key-to-disk system such as the Data 100 data entry system, you are buying specific hardware for a data entry chore oriented towards the data entry profession. A microcomputer data entry system carries with it all of the same benefits of an intelligent data entry system, but running on a general purpose piece of hardware that can be used for word processing, or running Visicalc, or other programs. Furthermore, the microcomputer-based data entry software, in general, tends to be more user-oriented and more oriented towards spreading applications out to the user rather than keeping the data entry function local to the data processing department.

HYBRID SYSTEMS

The most exciting systems on the horizon are those which are combining the online and offline capabilities which we have been describing; that is, systems that may operate sometimes online or offline depending on the desired properties of the system. Imagine an office of your company that has an APPLE or IBM Personal Computer or an HP125 sitting in it, perhaps performing word processing or financial modeling, but which can also be used as a data gathering station. What are some of the kinds of hybrid applications that we could make use of with this configuration?

Perhaps the most trivial example of a hybrid application is in performing a function we call error turnround handling. Imagine an order entry application that may have a variety of forms associated with it, perhaps representing an order header, repeating line items, and repeating partial shipments for each line item. Imagine that these transactions are gathered using the standard offline data entry software and are put into a batch, are checked and perhaps listed, perhaps batch totaled and are finally transmitted to the transaction processing HP3000. Clearly, there are going to be some database semantic errors which cannot be checked on the local microcomputers: credit limits might overflow, certain products need to be checked against the major database, and so forth.

The conventional solution for correcting these kinds of errors is to generate an error listing, send it back to the data entry operator, and have it rekeyed in the next batch. Our software provides an error turnaround mechanism by which the data that is sent to the transaction processor can have a tag put on each data segment that shows where it came from in the original source batch. When it detects an error in a record, the transaction processor can strip off this header fragment and write it to an errors file and gather up a file which represents all of the errors that were found. For example, out of 100 sales orders entered, three of them may have database failures which require further information to be entered. Using our communications software, this errors file can be sent back down to the microcomputer where a utility that we provide automatically runs that errors file against its original batch and creates an errors batch with only the three error records. Furthermore, the errors are now marked with the database codes such that the user can simply take them back into data entry system where a "Correct" command will automatically lead them to the field in error with an error message reported from the remote computer. This means that error turnaround information is not rekeyed but is simply sent back down and automatically creates a new errors-only batch. When this batch is corrected, it can then be sent up again to the transaction processor for reprocessing.

A more exotic second example which can accomplish the same function is as follows. Imagine the same data entry application except that at transmission the transactions are going into the transaction processor as they are being sent by the microcomputer. Thus, a record segment of an order is transmitted and immediately checked against the database in real time. As an error is detected, the transaction processor immediately brings the form back up on the remote computer and instructs the operator to repair it immediately. Notice very carefully that the very same form is being used in the online as well as the offline case to gather, correct, maintain, and modify the data. The operator learns only one interaction protocol whether operating in an offline or online environment. Using this mechanism errors can be corrected as soon as they are found.

The most interesting concept for hybrid systems involves an interactive transaction processor. Imagine a transaction processor which lives on the HP3000 and which can recognize transactions coming into it. Those transactions can either be in a batch or one at a time. In this type of a system, the software could be designed in such a way that depending on the preference or the needs or the requirements of the application a user may be offline or online. Again, consider our order entry example. An operator types in orders offline for an hour or two in the morning when suddenly a high priority order comes into the office. Using our hybrid system the operator can immediately put the microcomputer into an online mode, invoke the transaction processor and enter one order. The order is then entered immediately. The operator then drops offline, and continues entering more orders in the batch. Later, the operator again makes contact with the remote computer and now invokes the very same transaction processor and dumps in the morning batch.

You can see from these exaples that there are an infinite variety of hybrid applications. The important thing to recognize is the fact that the difference between online and offline is a distinction that has been made out of historical necessity. Microcomputer-based systems suggest that the microcomputer is the real interface to the user's information network. How the microcomputer chooses to handle the transactions — that is, whether online or offline — merely becomes an applicatin dependency or a priority dependency. As time goes on, we will see that the importance of these hybrid systems will make for very user friendly systems for which the user need not get involved in many of the details that we now consider essential.

CONCLUSION

I remember as a child getting up early in the morning to watch physics programs on 1950-style educational television. At that time the concept of educational TV was that a television camera would be placed in a studio that looked like a classroom. A teacher, with a desk in front and a blackboard in back, with a pointer for the blackboard, would make all of the motions of a teacher teaching a class and the TV was merely an observer to the classroom. Compare that technology with the technology of a "Nova" or a "Cosmos" in which suddenly television is recognized as being a medium with its own powers of communication, with properties far different than the classroom teacher.

We have witnessed this very same phenomenon in the growth of microcomputer software. Microcomputer software has simply assumed the role of minicomputer and main frame software without our examining the special attributes of the new medium. Small computers add something dramatically new into the computer picture. It is important for us to recognize those things that they do well, not so that we an exclude certain types of software applications from a small computer, but to figure out how we can use the particularly strong capabilities of the microcomputer to work in a friendly, compatible environment with our existing communications networks.