A DATABASE FOR DISTRIBUTED MANUFACTURING CONTROL

Alan Skinner, David Hutchison and Bruce Armitage*

This paper describes a database used for the control of a small manufacturing cell. The data used to control the manufacturing operation is distributed around the cell. Control is achieved by producing a piece of data (eg, the detection of a workpiece) at one node and duplicating that data on a second node. Access to the data is made via a Local Area Network (LAN). The user interface to the database allows the cell components, data variables, and the data flow to be defined. Once the cell has been defined any software required by remote nodes can be downloaded and the system set in operation. The work described in this paper adds an easy to use interface to a proprietary communications network. This greatly reduces the level of understanding of the communications protocol required to set up a working system and also simplifies the programming of network nodes.

1. Introduction

The use of Local Area Networks in manufacturing allows control functions to be distributed across an application. Fieldbus networks (Wood, 1986; Armitage *et al.*, 1988; Morris, 1990) in particular are being developed for low level communications in a manufacturing control environment. The distribution of functions means that a method must be found to manage the data requirements of the functions. One method of doing this is the use of a real time distributed database. A proposed Fieldbus network is (FIP, 1988), which is a development by a group consisting mostly of French companies. As well as providing the communications between the distributed functions, FIP provides a management system for distributed database.

As part of a project at Lancaster university to study the performance of Fieldbus networks (Armitage *et al.*, 1990) a small work cell has been set up. The cell functions are distributed across a number of network nodes.

*School of Engineering, Computing, and Mathematical Sciences Lancaster University, UK

The nodes each consist of a microcontroller to control local functions plus a communications interface to access other nodes. The local area network used, Intel's BITBUS (Intel, 1986), provides a microcontroller with a built in SDLC–based (IBM, 1984) serial communications controller.

The workcell had been converted from traditional star–wired control to network control (Skinner and Armitage, 1989) in order to utilise the flexibility of distributed control. It was also thought desirable to build a Fieldbus type application layer above the network. It was decided to follow the FIP practice of viewing the system as a distributed database, and to this end a database system has been developed. This database contains data relating to network nodes and the communications required to control the manufacturing application. Although this description of the networked application is held on a central computer, the data required by the applications control functions is distributed throughout the network nodes. The user defines network and its traffic in terms of logical names for devices and variables, and specifies the frequency at which transactions involving the functional data should occur.

2. Database Requirements

The database must allow the user to enter, review, and edit data relating to the network nodes and the cell components connected to these nodes. This data should consist of a means to identify easily components and functions. To allow for flexibility some nodes may be capable of supporting different functions. Various functions can be supported by downloading software to the node. Therefore the software running on a node for a particular application should be specified in the database. The control functions are distributed across the network. Some node functions may stand alone and require no interaction with other nodes. However in some cases a function may be distributed across more than one node, therefore a variable generated at one node (e.g. liquid level in tank) may need to be made available at a second remote node (e.g. pump drive motor). Control of the cell is then achieved by moving variables from one node to another via a communications medium, as illustrated in Figure 1.

The distribution of control functions means that a method of defining the movement of control variables over the network must be provided. This is achieved by a database record listing data transactions that make up the network traffic for an application. The data to control the cell is stored in two record sets, namely the Node Records and Message Records.

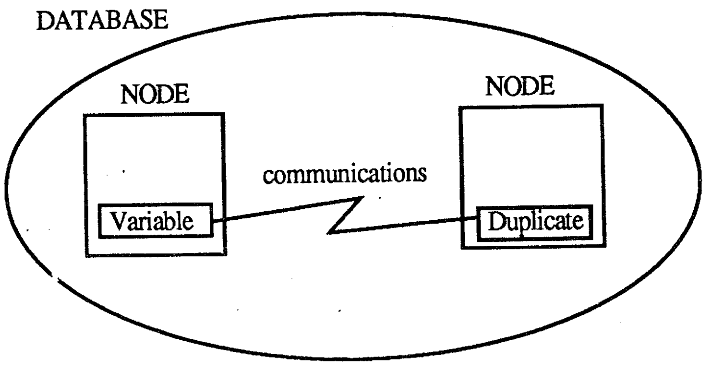

Fig. 1. Duplication of database variables

2.1. Node Records

The Node Record, as shown by the example in Figure 2, is split into two sections. One section holds data relating to the node and its associated software, the other is for information about the node variables.
The Node Information section lists a logical name for each node related to a physical network address. A brief description of the cell component function is included as is a filename for the software running on the node. The Initial Task Descriptor (ITD) is a requirement of the LAN Communications executive.

In the Variable Information section all variables accessed by the node are listed. Variables generated by the node and duplicates of remote variables required by the node are listed. This list relates logical variable names to physical addresses in the node Random Access Memory and the size in bytes of the variable.

LOGICAL NAME	WS2
ADDRESS (hex)	20
FUNDATION	work station 2 controller
DOWNLOAD FILE	\alan\rig\rac\s\ws2 \ws2.hex
FIRST ITD (hex)	fff0

VARIABLE	**ADDRESS**	**LENGTH**
start	300	1
buff	301	1
p_set	302	1
c_adrs	303	1

Fig. 2. Node Record Example

2.2. Message Records

The Message Records, (see the example in Figure 3), contain a series of data transactions that must be undertaken to control the cell. These are stored in terms of the logical names for nodes and variables. They list the source location of the data and destination location of the duplicate data. The period in ms between successive executions of the data transfer is given for each transaction.

No.	SOURCE	SINK	PERIOD (ms)
1	WS2.buff	WS1.c_2buf	200
2	WS2.p_set	ADR.c2pset	200
3	ADR.adrs	WS2.c_adrs	500

Fig. 3. Message Record example

3. Use of the Database

3.1. User Interface

Two versions of the database software exist. Both run on IBM PC's but one uses the interface functions of Microsoft Windows. The MS–Windows version is described here. A typical screen display is shown in Figure 4. The display shows the nodes as icons and the data transfers between nodes as lines. Nodes and Variables are identified by logical names. A node can be selected and expanded to show either the Node Information or the Variable Information section. The system is menu driven and the main functions are described below.

- The *File* option allows network applications to be loaded from or stored on disk and new applications to be designed.
- The *Edit* option allows the user to add a node to or delete a node from the network or to amend the Node and Variable Information fields.
- The *Messages* command is used to display the current Message Record list. Individual messages may be added or deleted, or the entire list cleared. The database display indicates messages in the list by a line between two variables. Messages may be entered into the list either from the keyboard or directly onto the screen by using a mouse. To enter a message directly the cursor is positioned over the source variable and the mouse button pressed, the cursor is then repositioned

over the duplicate variable and the mouse button released. The user is then asked for the period between successive transfers for this variable. This data is then automatically added to the message list.

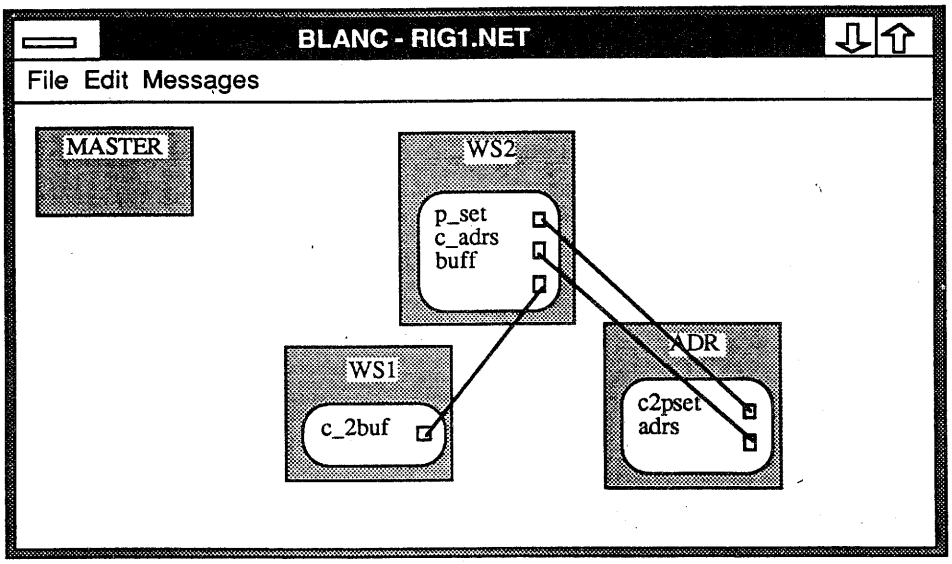

Fig. 4. A typical screen display of the database

3.2. Network Interface

The complete database control package consists of five programs, four of which run on the host PC. The database editor is described above, the other programs use data entered via the editor to control the workcell.
The Kill routine is used to halt an application running on the network; it uses the database to find the physical addresses of all nodes connected to the network and resets them.

The Download routines also access the network nodes. This routine uses the Node Record Download File field to locate the software needed at a node for an application to run. The software is downloaded over the LAN to the node located at the address stored in the Physical Address field.

The final PC based program is the Run routine. This uses both the Node Record and Message Records to produce a network traffic list. This is a list of messages coded in a form that the Intel BITBUS LAN communications software can understand. A description of BITBUS follows to aid understanding of the Run routine.

3.3. BITBUS Local Area Network

The core component of a BITBUS network is Intel's 8044 BITBUS Enhanced Microcontroller (8044 BEM) (Intel, 1986) which provides a microcontroller with built in communications executive. BITBUS is a single–master multi–slave network and is configured in a bus topology, as shown in Figure 5.

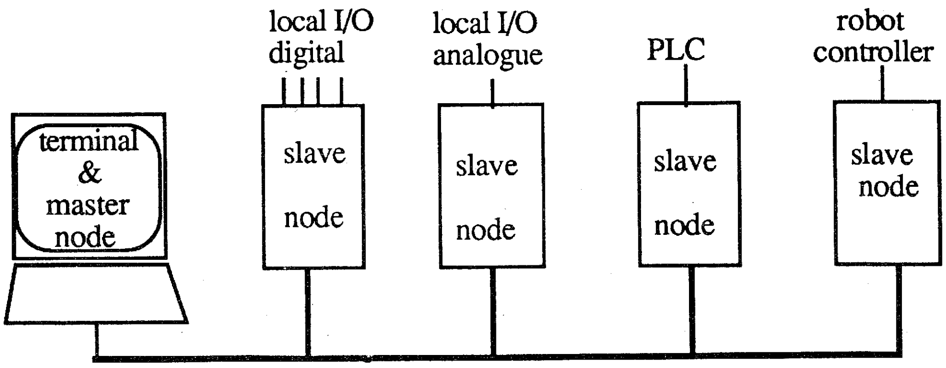

Fig. 5. BITBUS Local Area Network

The bus master which controls all access to the network typically resides in an IBM PC, the host PC being known as a BITBUS extension. Slaves may be connected to any application from simple on/off transducers to further extensions.

BITBUS uses a three layer architecture, corresponding to the physical, data link and application layers of the ISO Open Systems Interconnection (OSI) Reference Model.

The BITBUS network can be physically connected in a variety of ways giving different numbers of nodes and lengths of network. These are summarised in Table 1.

Mode	Repeaters	Nodes	Length	Speed	Wires
synchronous	no	28	30m	0.5 – 2.4Mbs	4
self_clocked	no	28	300m	375Kbs	2
self_clocked	no	28	1200m	62.5Kbs	2
self_clocked	yes	250	>1000m	62.5/375Kbs	4

Table 1. BITBUS physical layer options

Signals are carried over a differential pair conforming to the RS485 electrical standard. In synchronous mode two differential pairs are required, one for data and one for the clock, the logical state of the data signal being obtained at the rising edge of the clock signal. In selfclocked mode with no repeaters only a single pair of wires is required, and data is transmitted using Non Return to Zero Inverted encoding to allow clock timing to be extracted from the bit serial data stream. The network can accommodate up to 250 nodes by using repeaters although this requires an additional signal pair to control the direction of data flow.

The BITBUS link layer uses a subset of the IBM Synchronous Data Link Control (SDLC) protocol, SDLC being a subset of the ISO HDLC protocol (ISO, 1984). The BITBUS messages are placed in the information field of an SDLC frame. All normal transactions commence with an information frame from the master to a slave node; if no information frame is available at the slave it is repeatedly polled by the master until one is returned, and the transaction is completed by an acknowledgement from the master to the slave. BITBUS message transactions are transparent to the user.

The BITBUS application layer provides the user with a multitasking environment. Information is exchanged between tasks by the use of BITBUS messages, using the message format shown in Figure 6.

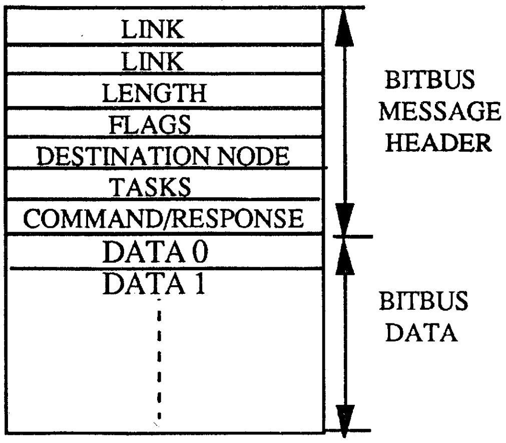

Fig. 6. BITBUS message format

Communicating tasks may be on the same node or on remote nodes. At this level BITBUS data transactions consist of an Order message followed by a Reply message. Order messages to tasks on remote nodes may only be sent by the network master node.

The BITBUS message is split into header and data sections. The header consists of six fields. The Link field contains 2 bytes of data used by the BITBUS executive and is not accessed by the user. The Length field contains the total length (header plus data) of the message. The current maximum message length is 20 bytes, the 7 byte header and up to 13 data bytes. The Flags field indicates Order or Reply messages, and whether the communicating tasks are on a node or a node extension. The Destination Node fields holds the physical address of the node receiving the Order message. The Tasks field holds identifiers for the communicating tasks. The Command/Response field is used in an Order message to indicate a particular message type, eg Read Memory; in a Reply message this field may be used to indicate an Order message request was successfully carried out.

The BITBUS based boards used in this application run a preconfigured version of Intel's Distributed Control Executive iDCX51 (Intel, 1986; iDCX, 1986), together with the BITBUS firmware iDCM44 (iDCX, 1986). The iDCX51 executive provides the multitasking environment whilst the iDCM44 firmware controls communications over the serial BITBUS link and over a parallel interface to an extension computer such as an IBM PC. The iDCM44 also provides Remote Access and Control (RAC) functions which provides a high level interface to the various read/write commands at a remote slave node. It is these read/write memory commands that are used by the database when constructing the network traffic list.

3.4. Database Operation

The communications protocol used is a single–master multi–slave type. Therefore each data transaction involves accessing the network four times. First the network master sends a read data request to the source node, and the node replies with the data. The master then sends a write data message containing the data to the destination node, which returns an acknowledging message back to the master, as illustrated in Figure 7.
Communications are controlled by a Master program running on a BITBUS board located in the database PC. The database Run routine produces BITBUS headers for a read data and a write data message for each transaction in the Message Record and stores them in the network traffic list. This network traffic list is loaded into the Random Access Memory of the network master node. The following paragraphs describe how a network traffic list entry is produced for one transaction.

The logical Name field of the Node Records are searched for a name corresponding to the source node name in the Message Record. Once the Logical Name is found the data in the Physical Address field of the Node Record is entered into the Destination Address byte of the read data message header. The Flags field is then filled, and both the source and destination tasks run on BITBUS nodes. The source task of the message is the Master communications program, the destination task is the BITBUS RAC task; this information is used to fill the Tasks field of the message header. The read data message of the transaction is a RAC Read External Memory message; the Command/Response code for this type of message is entered into the appropriate field.

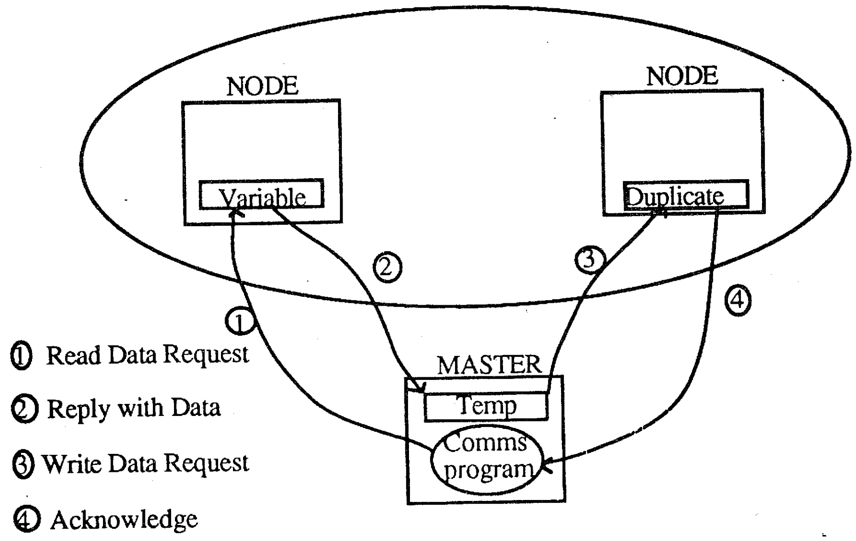

Fig. 7. Messages involved in a data transaction

The first two data bytes of the message contain the address of the memory location to be read. To fill these bytes the Variable Information section of the Node Record is scanned for a Variable Name corresponding to the Source Variable Name in the Message Record. Once a match is found the variable address can be entered into the message. The entry in the Variable Length field is used to calculate the length of the BITBUS message and this is entered into the Length field of the message header.

This process is repeated for the destination node and variable in the Message Record. The Period entry of the Message Record is used to set a time flag indicating when the transaction should take place.

Once the conversion of all messages in the Message Record list has been carried out the network traffic list is loaded into the master node. The system is now ready to operate. The run routine resets each node on the network and then uses the First ITD field of the Node Records to determine whether software loaded to the nodes starts on reset. If any software does not start on reset a message is sent to the node to begin its task.

The Run routine then interacts with the Master routine either to display measurements of the system performance or to stop the network operation. The Master routine scans the time flag of each entry in the network traffic list. If the transaction should take place the read data message is loaded into a transmit buffer. The message is transmitted and the master then waits for a reply. The variable data from the reply is loaded to a buffer that contains the header and first two data bytes (variable address) of the write data message. This message is send to the destination node in the Message Record. When the reply to this is received the time flag of the network traffic list is incremented by the message period and the Master routine moves on to the next entry in its list.

4. Application

The application used to demonstrate the database is a small manufacturing work cell. The cell consists of a conveyor and two in–line work stations (Tizzard and Cripps, 1986). The first work station is used for a manual operation. At the second work station a robot performs a simple profile following task.

4.1. Network Hardware

To control the cell using BITBUS one master node and three slave nodes are used. The master node is an iPCX344A BITBUS IBM PC Interface Board (Intel, 1986). The slave nodes consist of an iRCB 44/10A BITBUS Digital I/O Remote Controller Board (Intel, 1986), a power supply, and an optically isolated interface to the station transducers. Each work station slave is responsible for controlling the pallets within the work station. The third slave node is used to interface with an ADEPT One robot controller.

4.2. Software

The Kill, Download, and Run routines are written in Microsoft C 5.1, and interact with BITBUS through Intel's Universal BITBUS Interface (UBI). The Edit routine is written using Microsoft C 5.1 and the Microsoft Windows

Software Development Kit. The software for the master and slave node is written in PL/M51. The robot profile tracing software is written in VAL II.

4.3. Operation

Each work station consists of two areas, a buffer store and a work area. The workpieces are carried along the conveyor on pallets and are stored in the buffer area until the work area is empty. The first pallet to reach thc buffer store is held there by a raised pneumatic stop. A microswitch detects the presence of a pallet at the buffer store. Movement of a pallet from the buffer to the work area is effected by the slave node briefly lowering the stop and allowing the pallet to proceed along the conveyor. Once the pallet reaches the work area it is raised above the conveyor and locked into position by pneumatic actuators. The required task can now be carried out on the workpiece. Once the task is completed the pallet is returned to the conveyor.

Figure 4 shows the nodes, control variables, and transactions used in this application–the three transactions shown in Figure 4 are carried out by the Master routine to control the manufacturing operation.
The first workstation slave (WS1) node carries out its task on the pallet and then monitors the *c_2buf* variable in its memory. This variable is a duplicate of the second workstation (WS2) variable *buff*. This variable indicates the status of WS2's buffer store. When *c_2buf* indicates that the buffer is empty WS1 releases the pallet and monitors to its own buffer for the next one.

WS2's variables *p_set* and *c_adrs* and the variables *c2pset* and *adrs* in the robot interface slave (ADR) are used to control the operation of the robot. At WS2 the variable *p_set* is set when a pallet is locked into position ready for the robot to work on it.

The robot slave monitors the duplicate variable *c2pset*, and when this is set the robot is signalled to start its task and *adrs* is set. Once the robot task is complete *adrs* is reset. The robot interface again monitors *c2pset*, until it is reset. This indicates that the pallet has been released and the robot slave can begin monitoring *c2pset* for the next pallet.

After setting *p2set* WS2 monitors the variable *c_adrs*. This is a duplicate of the ADR *adrs* variable. Once this variable has been set and reset the pallet can be released and *p_set* reset. WS2 monitors its buffer for the next pallet.

5. Results

Two sets of results were obtained. For the first set the Master routine scanned through the network traffic list every 50 ms and carried out data transfers when necessary. For the second set of results the time fields in the network traffic list were ignored and the Master repeatedly scanned the list carrying out each data transfer at every scan.

5.1. 50 ms Scan

The master node waits for an interrupt signal which occurs every 50 ms. Upon the interrupt it scans sequentially through the message table. If the current time corresponds to the time for a transaction to occur the data is read from its source and written to its destination, and the time field for a completed transaction is updated by the transaction period. A timer is started by the master when a transaction to be performed is encountered. This is used to measure the time taken to process the transaction. It also measures the time from the send message request by the master to receipt of a reply from the slave. Both measurements are cumulative for one 50 ms window, ie. for two transactions the total time to process two transactions and transmit and receive replies to two message is measured.
After all the entries in the message table have been tested a message is sent to the PC. This message contains the number of transactions performed in the window, the time spent processing the transaction data and the time taken to transmit messages and receive replies. This data is shown in Table 2.

MSG/WINDOW	FREQ	AVERAGE ACTIVE TIME (ms)	AVERAGE COMMS TIME (ms)
0	1113	0.39	0.00
1	725	5.66	0.28
2	97	10.94	0.55
3	0	0.00	0.00
4	0	0.00	0.00
5	0	0.00	0.00
>5	0	0.00	0.00

Table 2. Message timings for a 50 ms scan window

The table shows the number of data transactions in each 50 ms Window; the frequency with which that number of messages occurred in each window; the average times for processing and actioning the transactions (Active Time) and for transmitting and receiving messages (Comms Time).

5.2. Continuous Scan

The rig timing results shown below were obtained without using an interrupt to start the scan of the transaction table. All transactions in the table were carried out during each scan (ie the time field was ignored). At the end of each scan the timing data was sent to the PC and the next cycle begun. This was done using the three transactions required to control the application and also with added dummy transaction to increase the network traffic. Table 3 shows the Active and Comms time as defined above for 3, 10, and 15 transactions.

	Times (ms)	
Transactions	**active**	**comms**
3	15.72	0.83
10	53.06	2.77
15	79.98	3.90

Table 3. Transaction timings for 3, 10 and 15 transactions

6. Conclusion

The work cell is run by a distributed control network with the operation coordinated by moving variables from one node to another over the network. The benefits of distributed control include reduced wiring costs, simpler software, and ease of expanding a system.

The addition of a database to control the application improves the flexibility of the system as new nodes can be added to the network and simply programmed into the database. Also different versions of software can be run on any slave node by changing a database entry.

Tables 2 and 3 show that the time taken to process a single transaction is approximately 5 ms. This suggests that for a control loop to function acceptably on the network it must not require updating at a rate greater than every 5 ms. Also, the greater the number of transactions required to control the application, the lower the frequency with which variables can be updated.

If ten transactions are required then it takes 53 ms to update all variables in the list; this means no variable can be updated more than eighteen times per second. This may be qualified by the fact that not all variables will be updated at each scan through the network traffic list (see the Frequency column in Table 2). In general, though, the greater the number of nodes on a network and the greater the number of transactions the slower the control loops on the network will be.

It should also be noted from Tables 2 and 3 that for this implementation the communications time is approximately 5 % of the active time. The communications time includes both the time the messages are on the network and the time for slave nodes to process messages. This suggests that the network speed, and to some extent the communications protocol used, are irrelevant to the overall performance of the application.

References

Wood GG, (1986): *Fieldbus, a developing low level industrial LAN standard.*– Proc. EFOC/LAN Amsterdam, pp.322–324.

Armitage BN, Dunlop GJ, Hutchison D and Yu S, (1988): *Fieldbus: an emerging communications standard.*– Microprocessors and Microsystems, v.12, No.10, pp.555–562.

Morris FJ, (1990): *IEC Fieldbus Performance.*– Birmingham: Advances in Measurement.

FIP Technical Description, (1988): Club FIP, Nancy, France.

Armitage B, Hutchison D, Merabti M and Skinner A, (1990): *Performance Characterisation of Local Area Networks in Manufacturing Applications.*– ACME Conference, Birmingham University, UK.

Intel, (1986): *Intel Distributed Control Modules Databook.*– Intel Corporation, Santa Clara, California, USA.

IBM Corporation, (1984): *IBM Synchronous Data Link Control General Information.*– GA27–3093.

Skinner A and Armitage B, (1989): *Manufacturing Control.*– Internal Report, School of Engineering, Computing, and Mathematical Sciences, Lancaster University, UK.

ISO, (1984): *Data communications – high level data link control procedures – frame structures.*– ISO 3309 International Standards Organisation, Geneva, Switzerland.

iDCX, (1986): *iDCX Distributed Control Executive Users Guide For Release 2.0.*– Intel Corporation, Santa Clara, California.

Tizzard GA and Cripps MD, (1986): *Low cost Hierarchical control of an assembly flow line.*– IFAC Low Cost Automation , Valencia, Spain, pp.271–279.