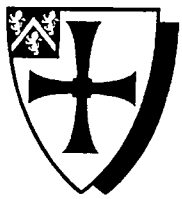
An application of an ethernet based protocol for communication and control in automated manufacturing

Edy Bertolissi

University of Durham



An Application of an Ethernet based Protocol for Communication and Control in Automated Manufacturing

Edy Bertolissi

Robotics Research Group,

School of Engineering.

Submitted for the degree of

Doctor of Philosophy

This thesis is dedicated to my family who have given me so much love and support throughout my life

Questa tesi é dedicata alla mia famiglia che mi ha incessantemente amato e seguito durante la mia vita

Abstract

The exchange of information in the industrial environment is essential in order to achieve complete integration and control of manufacturing processes. At present the majority of devices present in the shop floor environment are still used as stand alone machines. They do not take advantage of the possibilities offered by a communication link to improve the manufacturing process. The subject of this research has been centered on the development of a simple, flexible and inexpensive support system for communication and control of manufacturing processes. As a result, a system with these features has been proposed and implemented on a simulated workcell. The area footwear manufacturing was chosen for modelling the workcell.

The components of the manufacturing support system were developed using an object oriented approach which allowed modularity and software reuse. In order to achieve communication between the components, a communication protocol was developed following the process defined in the rapid protocol implementation framework. Ethernet was selected for implementing the lower levels of the protocol. Java, a new object oriented programming language used for the implementation of the system, showed that it could became a promising language for the implementation of manufacturing applications. In particular the platform independence feature of the language allows the immediate porting of applications to systems with different features. The manufacturing cell simulation had shown that the times associated with the manufacturing support system operations are compatible for its use in applications where the response times are in the order of one second.

Acknowledgments

I would like to thank my supervisor, Dr. Clive Preece, for his advice and support throughout the three years over which this research has been conducted. I would also like to thank Mr. David Reedman for his invaluable advice on the manufacturing aspects of the project.

I am also grateful for all past and present members of the School of Engineering who have helped provide such a pleasant research and social environment, and especially Mr. Trevor Nancarrow for his assistance with the computing equipment.

I would like to thank very much my family for their parental support during my time at the University.

I would also like to thank all my friends who supported me during the these three years and, particularly, Marco, Nobuko, Carlo, Max, and all the other wonderful people of Graduate Society.

This research project was funded jointly by the EPSRC (Engineering and Physical Sciences Research Council) and BU Ltd. (formerly the British United Shoe Machinery Co.).

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Durham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Contents

List of Figures

Chapter 1

An Introduction to Manufacturing Communications

1.1 Introduction

Recent times have seen the dissemination of information technology outside the office environment. Computers have been introduced in the area of manufacturing planning and management, and more recently in the domain of factory control. A rapidly changing market which requires prompt development and commercialization of new products forces companies to rationalize all their operational sequences. The ultimate goal is to achieve a complete integration and control of manufacturing processes, material inventory, sales and purchases, administration, and engineering design. The use of a computer communication network over which the information necessary for process interaction and coordination may flow is the key factor for obtaining high degree of flexibility and productivity in a company.

The needs of sharing information are particularly important in the case where production of small batches is required. In this case the system has to be able to quickly adapt in order to fulfill the new production needs. In this environment a communication link would be ideal in order to coordinate the activities of the manufacturing devices and provide them with the information required to be able to promptly adapt to the rapidly changing production needs. A communication

link could be beneficial even in the areas where the employed manufacturing devices consist essentially of stand-alone machines, which do not need of any type of interconnection. Greater flexibility, more accurate monitoring of the operations involved in the production area, increase in the automation of the process, and reduced idle times are some of the possible advantages. Recently the development of the Internet opens to devices which embed communication capabilities a complete new range of possibilities such as remote diagnosis or software updates.

Many of the advantages of the use of communication among devices do not lie in direct cost reduction of the plant, but in areas such, improved quality, faster delivery and improved information flow.

Although the technology of computer networking is mature, its specific application in manufacturing environments has failed to gain wide acceptance. A large body of research exists in the general area of network control and communications, addressing the problems of modeling and performance evaluations as well as designing flexible system for accommodating future growth. However one of the major obstacles to system integration is the incompability between equipment produced by different vendors. In the past the attempt for defining a standard for manufacturing communications led to the development of MAP [Gen87]. However this complex protocol failed to become an accepted standard leaving the incompatibility problem unsolved. One of the main reasons of its failure can be located in the cost factor which made its implementation uneconomical for small applications.

1.2 Research Objectives

In recent years due to the reduction in cost of electronic devices and microprocessors a larger number of electronic systems have been integrated inside industrial machines with a consequent increase in their level of intelligence. Lately, embedded PCs, computers specifically designed to be used in manufacturing environments, are becoming common inside manufacturing devices, allowing greater flexibility and increasing the potential of the systems.

Small companies which use a limited number of manufacturing devices would benefit from the introduction of systems for the control and monitoring of manufacturing processes, but they require simple and cheap solutions in order to justify the installation of such systems in their plants.

In order to expand the beneficial influence of interconnected production systems the objective of designing and developing a integrated communication system for the next generation of industrial devices was defined. The task of implementing a manufacturing control system to allow an effective control of the manufacturing environment was also set as a target of the research.

The project aimed to learn from the lesson of MAP which showed that a standard cannot be built artificially, but has to rely heavily on well known, cheap, proven technologies in order to have a chance of becoming accepted. Moreover integrated manufacturing support systems have to be simple to understand and use, and they have to provide flexibility in order to be adaptable to the different needs of the single production plants. Therefore the development of a simple, cheap and flexible system was placed as the underlying theme behind the development of this research project.

In particular footwear production was chosen as an area where such a system could show its advantages. In this type of industry, until not long ago, all the industrial machines were stand alone devices which required a considerable amount of human intervention in order to carry out their tasks. However this is changing and at present there are several examples of devices which can carry out operations automatically with limited human intervention. The possibility of connecting them using a communication system would offer them the advantages outlined above.

This research project is targeted at the design of a novel approach for the development of an integrated communication and control system for manufacturing applications. It combines together in a new way a broad range of disciplines and state of the art technologies in order to achieve coordination and communication at the shop floor level. In particular the overall project provides a original framework which could be easily adapted to a wide range of industrial contexts.

The structure of the thesis can be considered as a logical division of the work

into several modules which are representative of the work carried out for the design and implementation of an integrated manufacturing control system.

1.3 Thesis structure

The rest of the thesis is organized in ten chapters. In chapter 2 the most important features which qualify communication protocols and local area networks are described. The ISO/OSI communication model is presented and it is compared with other alternative protocol approaches. In chapter 3 the problem of local area networks for industrial automation is addressed. An overview of the currently available standards is presented. The advantages of using Ethernet, a network usually used for office communication, in industrial environments are highlighted, and a study of its performance is presented. Then in chapter 4 the specific problem of automation in the area of footwear industry is reported. The traditional production process is described, and a description of the new trends in the area follows. In particular two manufacturing cells which can benefit from a communication system are described. A preliminary study for the implementation of a novel industrial process for electrostatical glue deposition on shoe components is presented in chapter 5. This device is of fundamental importance for the implementation of one of the two manufacturing workcells described in the previous chapter. Chapter 6 presents a description of the computer support system necessary for the proper operation of a manufacturing cell. The functions of the database system, the monitor, the scheduler and the dispatcher are highlighted. In particular a novel heuristic for the solution of the scheduling problem in a non blocking shopfloor system is presented. Chapter 7 gives a full description of the design of the control system for the manufacturing workcell. Chapter 8 describes a novel approach for the design and the implementation of communication protocols. This framework is applied to the specific problem of implementing a communication protocol for connecting manufacturing cells in the footwear environment. The description of the implementation of the system according to the design presented in the previous two chapters is reported in chapter 9. Java, a promising new object oriented language which

has the important feature of being platform independent, has been selected as the implementation language. A description of its advantages and limitations in the area of manufacturing applications is reported. Then the implemented system, and in particular the performance of Java, is assessed in chapter 10. Finally in the last chapter the advantages and disadvantages of the implemented ideas are discussed and suggestions for future work are given.

In addition to the main body of the thesis there are four further appendices which provide information on some aspects of the research carried out.

Chapter 2

Communication Networks

2.1 Introduction

A network architecture and its set of rules govern the connection and the interaction of the network components: they include physical structure, data format, and protocols and logical structures for the functions which provide effective communication between data processing systems connected to the network [Fre88].

The existing communication systems can be classified into two architectures [Ben93]:

- Closed System Interconnections (CSI);
- Open System Interconnections (OSI);

The CSI architectures are local networks in which all the components come from and are designed by the same vendor. The user is therefore forced to buy complete solutions and future extensions from one vendor. If devices of different vendors have to be connected, special solutions, usually expensive and complicated, have to be customized.

The OSI architectures try to develop vendor independent architectures, allowing communication between application systems which are usually not compatible with one another. This requires extensive standardization work which concerns the

communication interface and the way in which the messages are exchanged.

In section 2.2 an overview of the ISO/OSI communication model is presented. In section 2.3 other non ISO/OSI communication standard are presented. Finally section 2.4 will give an overview on the main qualifying features of local area networks.

2.2 The ISO/OSI model

In 1983 the International Organization for standards defined its Open System Interconnection (ISO)/OSI reference model (ISO 7498) [Int84]. OSI itself is not a standard but offers a framework to identify and separate the different conceptual parts of the communication process. It introduces a conceptual model for communication which is similar to the different levels of operating systems, where the operations have different abstractions, ranging from machine code and assembler programming to high level languages and applications. The multilayered structure of an operating system is essential for providing services, while hiding the implementation details from the higher layers.

Seven functional layers are defined in OSI (see fig. 2.1). They are traversed one by one by the message either top down or vice versa during the communication. Every layer provides services to the next higher layer and uses services of the next lower layer to execute its task. Therefore each layer exchanges information only with the adjacent ones [Tan88]. OSI service calls are similar to operating system calls: the requesting layer passes data and parameters to the layer below and it waits for an answer ignoring the details of how the request is carried out [OP92].

The layered approach ensures modularity and the facility for networking software to be upgraded without affecting the other layers of the communication system. The modularity, as already mentioned, makes it possible to use multivendor hardware and software in the same system.

Modules located at the same layer but in different positions in the network are called *peers*; they communicate via *protocols* that define conventions for exchanging information between two users of the same information [RD89]. The services are

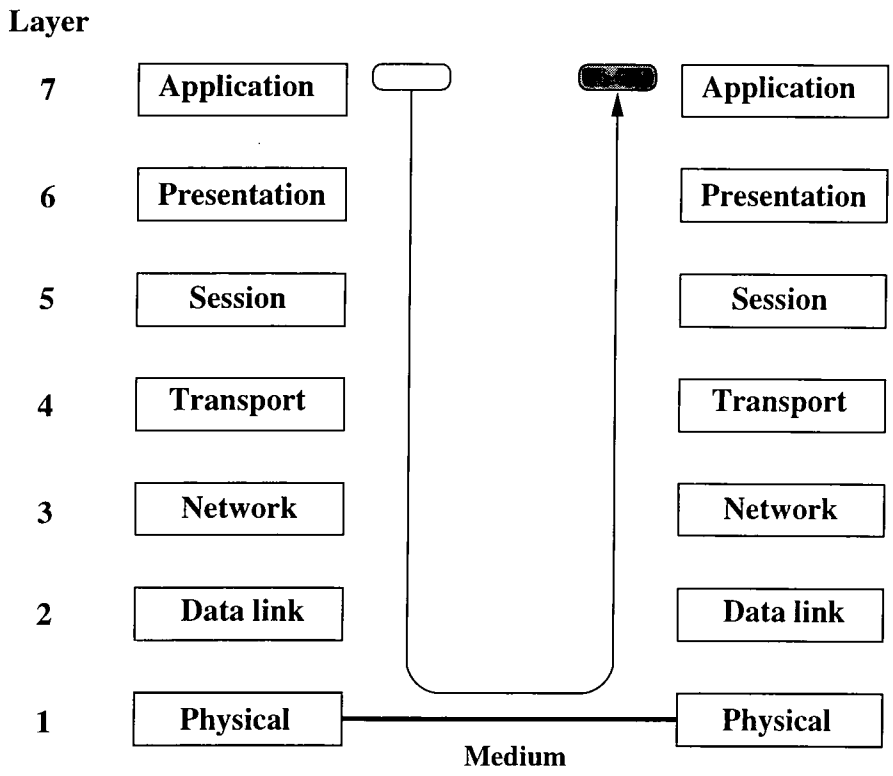


Figure 2.1: The seven-layer OSI reference model.

strictly separated from the protocols (the actual implementation).

The layers defined in OSI are the following [BDBL91] [Har85] [Tan81]:

- *physical link layer*: this layer is the lowest layer of the model and is concerned with all aspects of the physical interconnection of the interface to the cable. For example the standards will define the mechanical aspects such as the type of connector to be used, the electrical signals levels used for transmission and reception via the cable, and functional and procedural aspects such as the type of handshaking to be used. The reference model does not extend down to the physical medium itself so does not include a specification of the type of cable to be used. The physical layer is the only real connection between two communicating nodes;

- *data link layer*: this layer provides the functional and procedural means to establish, maintain and interrupt a data link over the network. It defines the way the serial data is organized and the detection and possibly correction of errors occurring during the packet transmission;

- *network layer*: this layer has the task to do the routing, the selection of a path through a network node, on which data transfer takes place. As the paths should be never overloaded the most efficient path must be found;

- *transport layer*: this layer provides an end-to-end communication control and it is the interface between the application software that requests data communication, and the external network. It has the responsibility of verifying that the data from one machine to another is transmitted and received correctly;

- *session layer*: this layer is involved with establishing the interactions between two users applications on different systems connected to each other by the network;

- *presentation layer*: the main function of this layer is to provide an independence to the user application from differences in the presentation of data. Thus differences in the way that computers talk to one another and the actual data can be resolved;

- *application layer*: this layer provides application specific protocols. It provides a set of services which can be directly called by application programs.

Protocols from layer one to four are called *network oriented protocols* whereas from layer five to seven are called *application oriented protocols.*

As data moves downwards to the physical layer two major things happen to it. Additional protocol-related information is appended to the original message (see fig. 2.2) which at the same time can be segmented into smaller pieces. This added information is specific for each layer and it is intended for the peer layer [McC88, OP92].

However OSI model is not exempt from criticism. Efficient implementation has not been explicitly addressed at all. Moreover some aspects of the implementation may require to use excessive resources of the system, and may produce performance degradation. Several times it has been pointed out that the division of the layers 4-7 is somehow academic. Even if there is the implicit requirement that all layers should be involved in each application process activity, OSI is usually

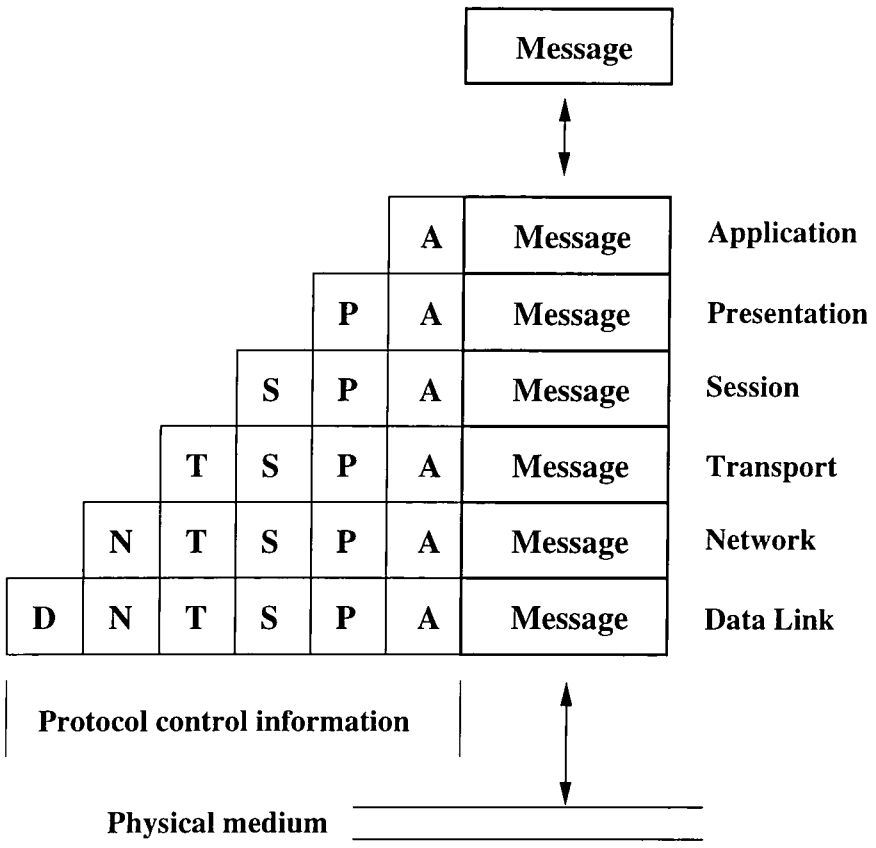


Figure 2.2: Data flow through the OSI layers. Protocol control information is added or removed at each stage while the message traverses the stack of transmission layers.

not implemented in its entirety, but computer manufacturers implement only the necessary layers avoiding all the features that are not necessary for the specific application. The most important faults in the OSI system is that it does not include important functions such as network management and security (data encryption): these omissions may cause the development of incompatible standards in future [McC88, OP92].

2.3 Alternative network architectures

While several manufacturers of computer communications systems and equipment vendors are evolving products following the directives provided by the ISO/OSI model, there is a variety of alternative network architectures and protocol layers

currently in use which do not conform to the model. It is unlikely that in the near future all of them will be abandoned, and in many cases it will be necessary to provide gateways between systems which comply with the ISO standards and the other ones. Alternative network architectures include:

- IBM, SNA;
- DEC, DNA;
- DARPA, TCP/IP;
- Xerox, XNS.

Among these, the most important one is TCP/IP developed in the early '70s by the US Department of Defense Advanced Research Projects Agency (DARPA), and included in the Berkeley Unix version 4.2 operating system. The Unix-TCP/IP has become the most wide-spread system used in local area networks for interconnecting computers. With the impressive development of the Internet it has become the *de facto* standard used for transmitting data at planetary level. The main advantage of this suite of protocols is that they are non-proprietary, and due to their popularity there is abundance of software written that relies on them. For these reasons it is unlikely that in future they will be replaced by other OSI conformant transmission protocols with similar features.

In fig. 2.3 [Hal92] it is possible to see a comparison between the TCP/IP layers and the OSI ones. The network layer is called the Internet protocol (IP); this datagram protocol addresses messages and routes them across the network and exchanges data between systems independent of the network topology and the media used. An additional Internet control message protocol (ICMP) provides network layer management and control functions. At the transport layer level it is possible to choose two different transmission protocols, the TCP protocol (Transmission Control Protocol) [Pos81], which implements a connection oriented service, and UDP (User Datagram Protocol) [Pos80], which provide a connectionless service. On top of this all the application specific protocols are developed, such as the *mail*

protocol [Pos82] for sending mail messages across the network, *file transfer* protocol [PR85] for file transmission, and the *telnet* protocol [PR83] for remote terminal connections. All application programs have the problem of establishing a common network representation. Unfortunately, not all computers agree on how data is represented. There are differences in character codes (ASCII vs. EBCDIC), in end of line conventions (carriage return, line feed, or a representation using counts), and in whether terminals expect characters to be sent individually or a line at a time. In order to allow computers of different kinds to communicate, each applications protocol has to define a standard representation. TCP and IP do not care about the representation. TCP simply sends octets. However the programs at both ends have to agree on how the octets are to be interpreted. A more complete description of the TCP/IP suite of protocols will be given in section 3.3.

2.4 Local area networks

Local area networks (LAN) allow the a group of independent computers spread over a local geographic area to share information and resources via an interconnection which permits high transmission and low error rate [Did89]. The elements which qualify a local area network are:

- network topology
- transmission media
- medium access methods

2.4.1 Network topology

The topology of the network refers to the way in which the computer and devices are connected to the physical cable. Today three forms of topology are used for communication networks [BDBL91, Fre88, Did89]. Star networks require that each system is interconnected with a central station or hub which controls the flow of

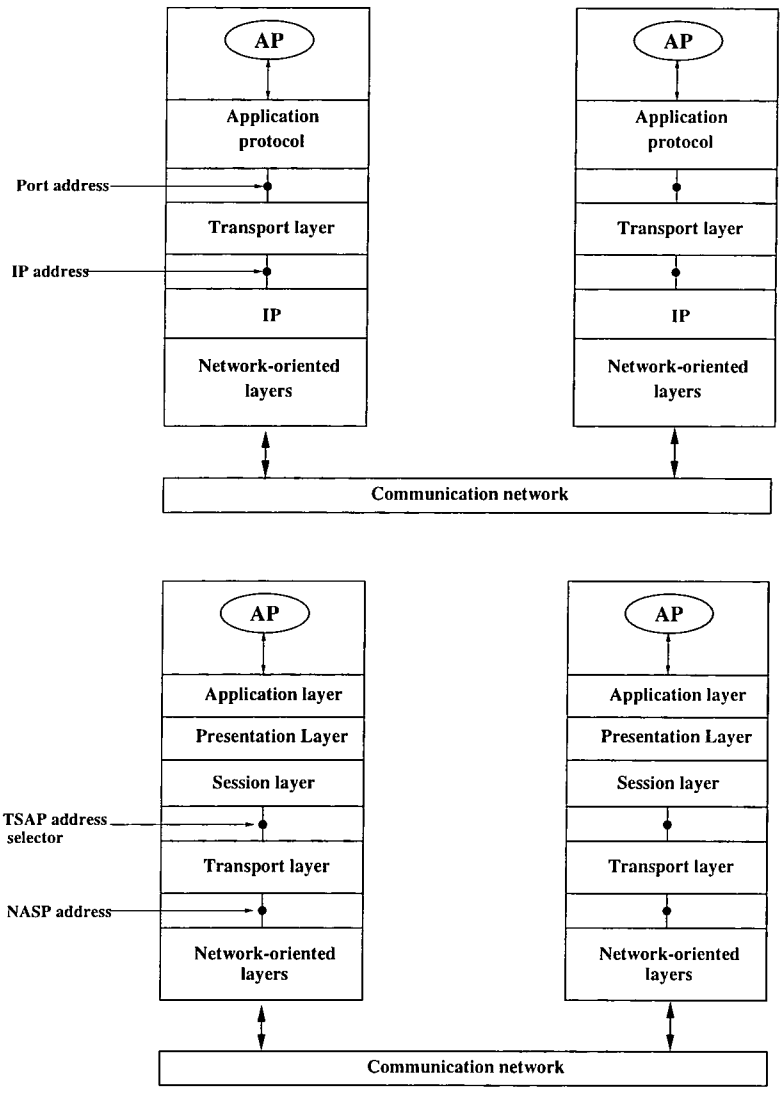


Figure 2.3: TCP/IP layers compared with OSI layers. (AP = application process, NASP = network service access point, IP = internet protocol, TASP = transport service access point)

information between the elements of the network. Bus networks employ a single cable to interconnect all systems. The communication between two systems can be established without involving a central controlling station. The total throughput capability generally decreases as the number of stations increases. Ring networks establish a loop by connecting each system or device to its neighbour.

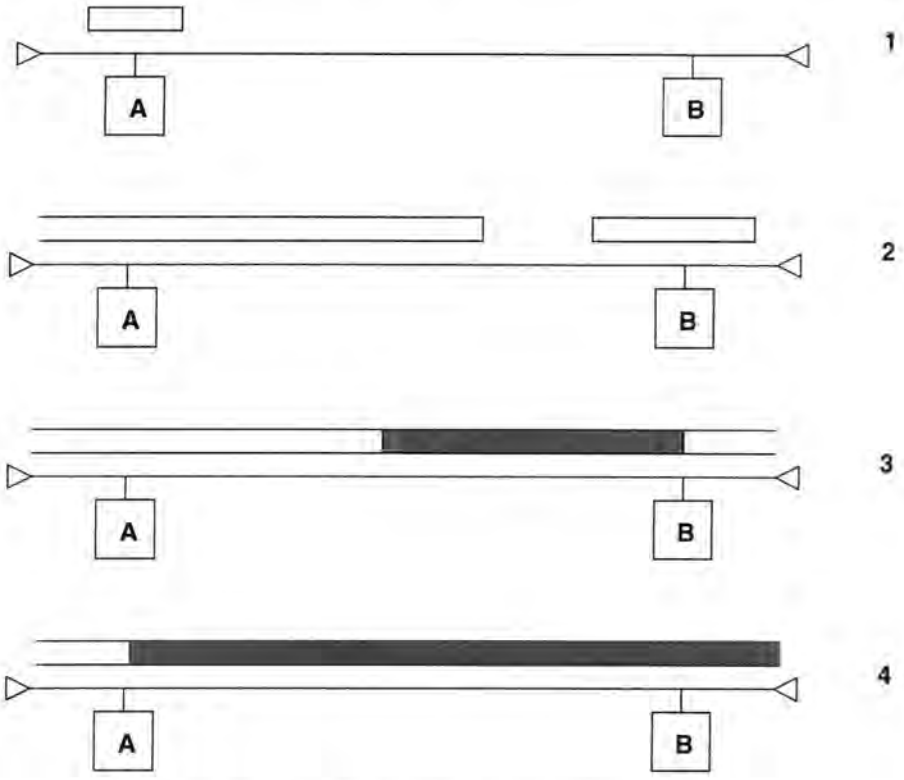


Figure 2.4: Ethernet operating principle.

2.4.2 Transmission media

LAN systems are serial bus systems, therefore the lines used to connect the nodes of the network are usually realized using one or two wires. The following transmission media are used [Fre88, Did89, Ben93]:

- twisted pair;
- coaxial cable;
- optical fibers.

Twisted pair cables consist of two insulated conductors which are twisted together. They are the least expensive solution, as they are cheap and easy to install, but they have a limited transmission rate and are susceptible to interferences. Coaxial cables surround the inner conductor with a dielectric such as polyethylene, and a coaxial tube of solid or braided metal surrounds the dielectric. Electrical interference is extremely low if the outer shield does not have gaps. Small coaxial cables are

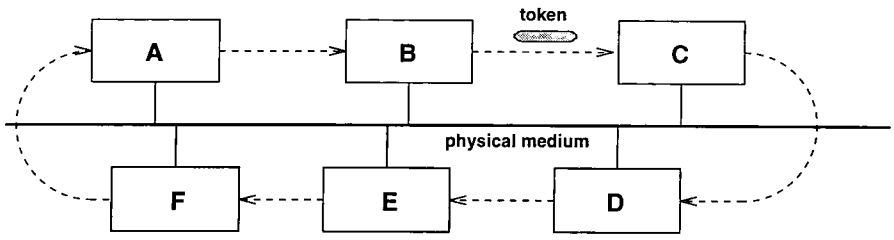


Figure 2.5: Token bus operating principle. The stations A-F circulate the token and can communicate when they hold it.

inexpensive, but low-loss ones are much more expensive, larger and less flexible. Fiber optic cable cores are made of glass or plastic. They have the high transmission speed and capacity. There are no problems with electromagnetic interference and they have very low error rates. They are generally smaller and more flexible than electrical cables. The costs associated with medium distance and long distance fiber optic-links are less than those for electrical cables.

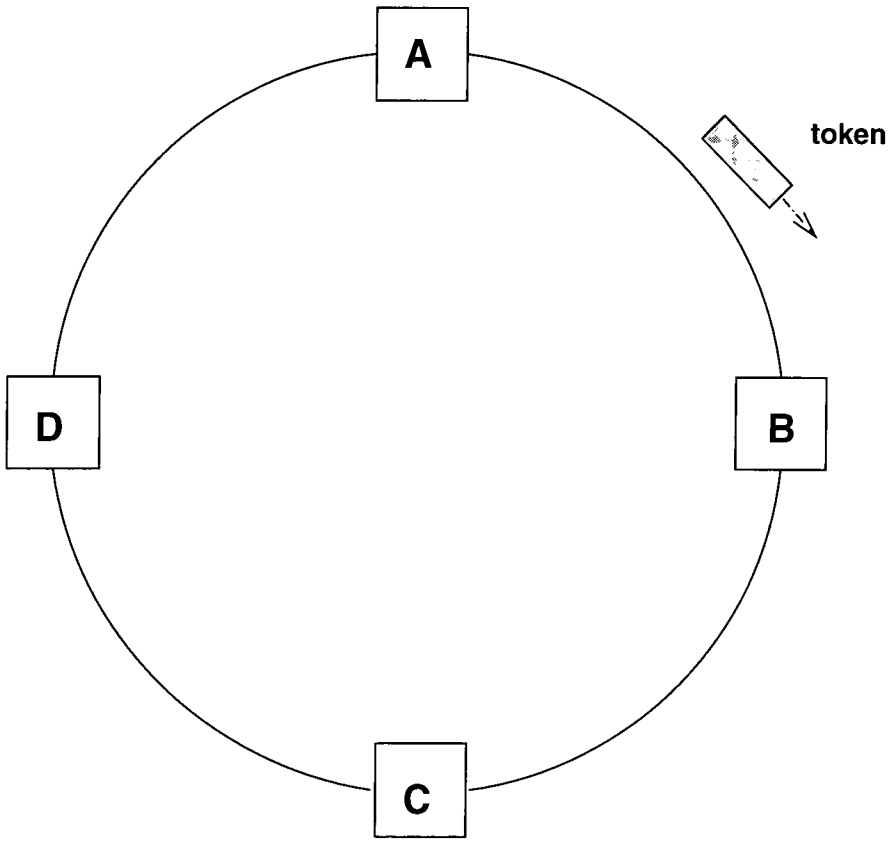


Figure 2.6: Token ring operating principle. The token is circulated among the stations. A station can send messages only when it holds the token. The stations connected identify and collect the messages directed to them and forward the others.

2.4.3 Medium access methods

Medium access is controlled according to rules which every station on the bus should observe for transmitting and receiving messages [Ben93]. The access methods which are usually employed in LANs are [Did89]:

- polling;
- random access;
- token passing.

If the polling technique is used, a master station interrogates sequentially the nodes connected to the network giving the access right to the transmitting media to the stations which need to transmit. The weak point of this architecture is the polling station: if it breaks down all the system fails.

In the random access technique there is no master station, and the access of the medium is determined by chance. Ethernet (IEEE 802.3) is the best known example of application of this method. When a station wants to transmit a message it waits until when the transmitting medium is free from traffic and it sends its message. If another station starts the transmission at the same time a collision is generated and the conflict is resolved by different algorithms (see figure 2.4). In the case of Ethernet after every collision each of the stations involved in the event use a exponential backoff algorithm which randomly decides when each system can try to use the bus again to complete its activity. If, on the new attempt of gaining control of the bus, another collision is detected the waiting time is doubled. If a station is not able to gain access to the bus after a certain number of attempts it assumes that the problem has a different nature and reports the situation to the higher layer [McC88].

No master station exists when a token passing technique is used. Token ring (IEEE 802.4) and token bus (IEEE 802.5) are two examples of token passing networks (see figure 2.5 and 2.6). Only one unit at the time can use the network for transmitting messages. A node is entitled to forward data over the medium only if it retains the token, which allows the holder to transmit up to a specified amount

of time.

Random access methods have a very good performance in lightly loaded, random, bursty environments. As there is no medium access control, the implementation of the system is simplified, and there is no need for monitor or token requirement functions. Stations can be easily added or deleted by simply activating or deactivating them. A station throughput depends only on the activity of the other stations when it attempts to transmit, while the number of the connected stations is not important. When the load increases, a delay in completing the operation appears due to the greater number of collisions on the bus. As the behaviour is not predictable on a small time scale, it is not possible to guarantee a particular level of performance. A fault at one of the stations does not compromise the functionality of the network. Only the bus itself is a weak point of the system, and if that is interrupted, all the system fails.

Token passing protocols require transmission capacity and time delays for protocol related information exchange. They are more expensive than the Random access solutions as they have to generate and process medium access control protocol data units. Token passing requires particular procedures for adding and deleting nodes. It is possible to predict the response time, and the token holding time can be adjusted to tune the performance. Token passing performance is a function of the number of active participants. Increasing the number of stations involved in the passing of the token increases the the delay and reduces the throughput. As the loading of the network increases the performance remains stable. In these systems if there is a fault in a single node, all the systems fails.

2.5 Conclusions

This chapter has provided an overview of the general features which characterize layered communication protocols. The seven layer approach defined by the ISO/OSI model was described and the functionalities of each layer highlighted. However not all the existing communication protocols follow the the ISO/OSI lay-

ered approach, and TCP/IP was described as the most important example which is not ISO/OSI conformant. The description of the qualifying features of local area networks were reviewed. A description of the possible network topologies, transmission media and medium access methods was proposed, underlying the advantages and limitations of the different solutions. Particular attention was given to the features, like ease of reconfiguration and robustness, which are most likely to be important in manufacturing applications.

The main aim of this information is to provide the background knowledge which is essential to understand the discussion on the different industrial local area networks which will be presented in the next chapter.

Chapter 3

Ethernet and Fieldbusses

3.1 Introduction

Methods for implementing control of processes and instrumentation have greatly changed through the years. Devices have been incorporated into processes which have made them more sophisticated. First the addition of simple electronic components, such as diodes or relays, made them perform higher level functions, and then with the introduction of micro-electronics, more and more sophisticated tasks have become possible. The introduction of microprocessors gave a certain level of intelligence to the devices which were able not only to perform their applications, but also to have the capability of giving at the same time other information on the performed tasks and their status.

The possibility of gaining full advantage of the characteristics of those intelligent devices forced the researchers to look at ways of interconnecting stand alone devices and replacing analog connections used to link low level devices, such as sensors and actuators, to a central unit. The idea of a digital communication link able to interconnect the devices with the control room, thus making plants more flexible, responsive and accountable made several companies propose proprietary systems for interconnecting their own equipment. In this way plants could be controlled in the most profitable way following the changing needs of the market. However this did not allow communication between devices from different manufacturers. This

problem lead to the the idea of creating a standard for linking equipment from different vendors in a transparent way, thus developing communications standards for industrial applications.

In section 3.2 an overview of the industrial communications systems currently available on the market is given. Section 3.3 gives an account on the advantages and disadvantages of Ethernet in industrial applications in comparison with the other communication systems. Section 3.4 reports the results of a series of tests for assessing the performance of Ethernet. Finally some conclusions are drawn in the last section of the chapter.

3.2 Fieldbusses and Sensor-Nets

The hierarchical approach of a communication system for manufacturing applications makes clear that different requirement profiles are needed at the different levels of the system in order to achieve the desired performance of the system.

At the bottom level of the manufacturing environment, and especially in process plants, networking begins with the so called sensor-nets, the multi-drop digital replacements of low level networks such as the 4-20 mA current loop. Digital smart chips are cheap enough to be added at a reasonable cost to simple sensors (e.g., photoeyes, proximity sensors, etc.), actuators (e.g., motor starters), and other components of an automation system that have traditionally been 'dumb'. The basic idea behind sensor nets is usually to have a bus (or trunk line) with power and two communication lines, and to connect not-so-dumb components to it (much like computers connect to an Ethernet).

These types of networks are designed to carry few bits of information and to perform at the best when they have to transmit short messages.

The advantages of using sensor nets in production plant can be summarized as follows:

- The wiring of an automation system with a sensor net is vastly simplified since there are single signal and power lines running around the system instead of

a large bundle of wires;

- Since the sensors and actuators have some processing power in them, they can listen to the signal line to determine when they are being queried, can construct response messages, and can handle the occasional conflict when 2 or more entities on the bus transmit simultaneously, etc. In addition to the communication features, these components could be programmed to announce other events such as a photoeye that senses that the light from its reflector is too low and thus may be out of alignment, a mechanical switch that keeps track of the number of times it is actuated and sends a request to be replaced when the number reaches some limit, and so on.

Hart, LonWorks, P-Net, Interbus-S and ASI belong to this category of networks.

It is possible to identify a separate set of sensor nets which are designed primarily to interconnect large numbers of digital sensors and PLCs, transmitting few bytes of information at very high speed, with features which make them suitable for hard real time applications. Examples of such networks are CAN, DeviceNet and SDS.

At the higher level there are fieldbusses, which are conceptually similar to sensor nets. Whereas sensor nets are meant for hooking up relatively simple devices such as sensors to controllers such as PLCs, fieldbusses are meant to allow complex subsystems to talk to each other as well. This means that fieldbusses can be used as sensor nets as well as more complex communication nets such as Ethernet. The amount of information which can be transmitted is greater then in the case of the sensors-nets and the number of services which can be performed is greater. Belonging to this category are fieldbusses such as FIP, Profibus, IEC/ISA SP50, and the one proposed by the Fieldbus Foundation.

Moving up through the hierarchy layers at the cell and process management level at present no standards are defined and networking usually relies on proprietary solutions. In 1980 General Motors created a task force in order to create a standard communication system able to link together all the computer controlled devices used in an industrial plant: this protocol was called Manufacturing Au-

tomation Protocol (MAP). Several versions of the protocol were released. However this ambitious project did not have the success that was expected. Incompatibility problems between different releases of the specification, extremely complex documentation, and high installation costs made the system less attractive and only few industrial applications had been developed. At the present the protocol seems to have been largely abandoned.

Above all there is the planning level which is only seldom interconnected with the layers below. At this level large packets and almost no real-time requirements prevail.

In the following section an overview on the most important sensor-nets and fieldbusses available on the market is given. In table 3.1 a summary of the main features of these networks is presented.

3.2.1 P-Net

Even though P-Net was awarded of the status of national standard in 1989 not many people knew of the existence of this protocol until short time ago. The majority of P-Net applications are found in the process industry environment, but there are some applications in discrete manufacturing plants. P-Net is a multi-master multi-net standard where the master sends a request and the slave returns a response. The multi-master access to the Physical Layer is performed using a virtual token passing principle between masters. A network can have up to 125 devices per bus and up to 32 masters connected per segment. The bit rate of the bus is 78 KBit/s which allows up to nearly 300 confirmed services per second. The maximum data length in one transmission is limited to 56 data bytes in order to avoid the overloading of the network produced by one node. P-Net implements layers 1,2,3,4 and 7 of the OSI stack and uses objects to define the I/O interfaces in the different nodes. These objects define real-time data as well as predefined function switches, diagnostic, maintenance data and error messages, The Physical layer is based on the RS-485 standard which allows segments up to 1200 m

Name	Medium	Access method	Max nodes	Data rates (bit/s)	Max length (m)
P-net	twisted pair	MS-T	125 (per seg)	76.8K	1200
ASI	twisted pair	MS	31	n.a.	100
Interbus-S	twisted pair, fiber	MS-T	64	n.a.	400
CAN	twisted pair, fiber	CSMA/CR	2^{11}	1M	500
SDS	twisted pair	CSMA/CR	128	1M	500
DeviceNet	twisted pair	CSMA/CR	64	125K-500K	500
Hart	twisted pair	DM	15	1200	1500
LonWorks	twisted pair, RF, coax, fiber, power line	CSMA/CA	2^{48}	38K-1.5M	1200
FIP	twisted pair	MS	256	31.25K-2.5M	1900
Profibus FMS	twisted pair	MS-T	127	9.6K-1.5M	1200
IEC/ISA SP50	twisted pair	several	256	31.5K-2.5M	1900
FF	twisted pair	MS-T	256	31.5K-2.5M	1900

Table 3.1: Physical characteristics of the leading fieldbus options available on the market. The details of the system tend to change quite rapidly as new features are added in order to make the product more attractive for the customers. The details about the meaning of the abbreviations used in the access method column are explained in the text.

long. There is no need of a specific chip as P-Net nodes can be implemented using standard single chip processors such as Intel 8051 or Motorola 6805 with standard driver software. This reduces the development costs of the nodes.

3.2.2 ASI

This protocol has been designed primarily for monitoring and controlling of two state devices at the bottom end of the automation hierarchy. It operates according to master/slave principle; only one master controlling up to 31 slaves is allowed on the bus which can be up to 100 m long. The master interrogates cyclically the slaves which can transmit or receive up to 4 bits of data. the maximum delay between two interrogations is 5 ms. As it uses 4 bits to transfer information it is not suitable to transfer analog values. The main advantages are the simplicity of wiring and the ease of installation.

3.2.3 Interbus-S

This is a deterministic protocol which has been developed in order to operate in almost real-time environment in accordance with the master-slave principle. It is suitable for communication among devices which need to transmit only few bytes of information. All the data from sensors and actuators in a network is summarized in one message which is sent simultaneously to all the connected devices. Interbus-S operates in a ring topology using the RS-485 specification for connecting up to 64 devices on the network. However the ring topology implies that if a station fails, the whole network goes down.

3.2.4 CAN, SDS, DeviceNet

SDS and DeviceNet are both similar but incompatible attempts from different groups to define the upper layers of the CAN protocol which in its original version defines only Layers 1 and 2. CAN is a true distributed system where a node broadcasts its data to all the other stations. The medium access algorithm on the bus is based on CSMA/CR (Carrier Sense Multiple Access/Collision Resolution). In the case of a free bus all the nodes which need to transmit start transmitting their collision resolution pattern (assigned at installation time and which reflects the importance of the node) which revokes the access to the bus to the low priority nodes (see fig. 3.1). The danger of this system is that some nodes may be locked out in the presence of persistent high priority node activity. As CAN had been developed for the automotive industry it has been designed to be used in difficult environments. Due to its origins it is not optimized for industrial applications; the length of the messages is limited (8 bytes). However it is provided with extensive error detection mechanisms. The bus supports speeds up to 1 MBit/s which makes it suitable for real time applications. SDS and DeviceNet are not intended to be alternatives to a complete fieldbus system, but are developed with the idea of providing digital communication between devices such as sensors, switches, drivers and programmable controllers.

3.2.5 Hart

HART is a well established low level fieldbus especially suited in process installations with a combination of analog and digital equipment. This protocol is able to carry a conventional 4-20 mA analog signal with digital data on a twisted pair without interference. However the transmission rate is low as it is possible to have up to 3 transactions/second when the two types of signals coexist and to 15 when the devices are used in digital only arrangements. For this reason it is suitable only for applications which require less than 10 messages/second (like the process industry where pressure, temperature and flow measures change slowly). The segments

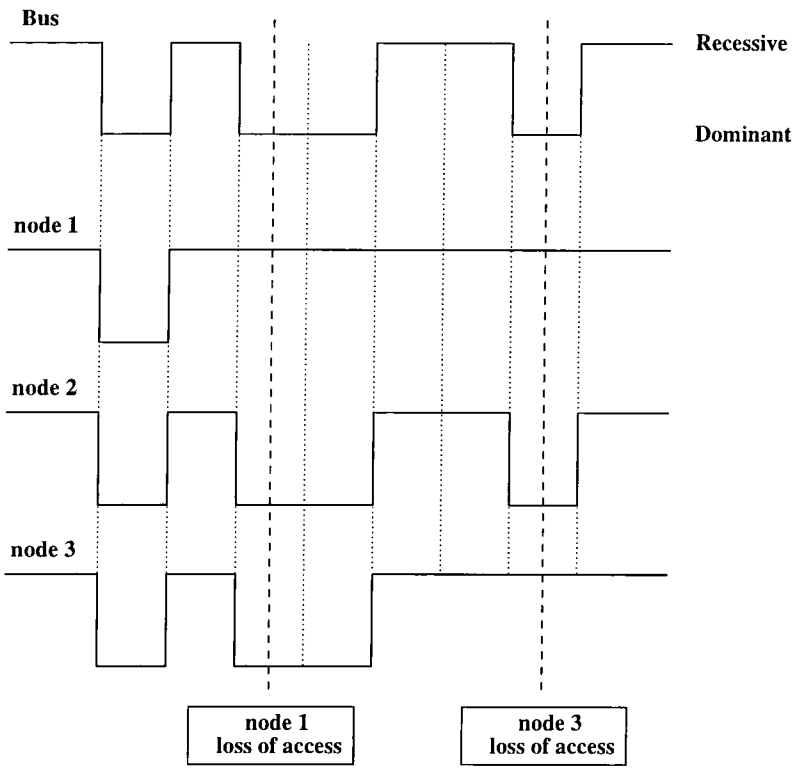


Figure 3.1: In this example three nodes try to transmit at the same time on the bus. All the nodes start to transmit their identifier which is used for the nondestructive bitwise arbitration. As when a node transmits a 0 on the bus the signal level is set to low, any other nodes attempting to set the line to high will notice the failure of their attempt of setting the level of the bus, and stop transmitting and start to listen to the bus. In this way the message is not corrupted if more than one station tries to transmit.

can be long up to 1500 m and the Data Link Layer uses a centralized dual master access system. The main advantage of HART is that it can be used in combination with old existing equipment which uses the 4-20 mA loop protocol.

3.2.6 LonWorks

LonWorks is a well accepted low level fieldbus based on a distributed architecture. Its main features include a wide range of supported transmission media (twisted pair, RS-485, fiber, power line and wireless), its low cost and the small chip size. The transmission rate depends on the medium and the transceiver design and varies from 5 KBit/s to 1.25 MBit/s. The protocol supports networks with segments using differing media. All the seven OSI layers have been implemented. The access

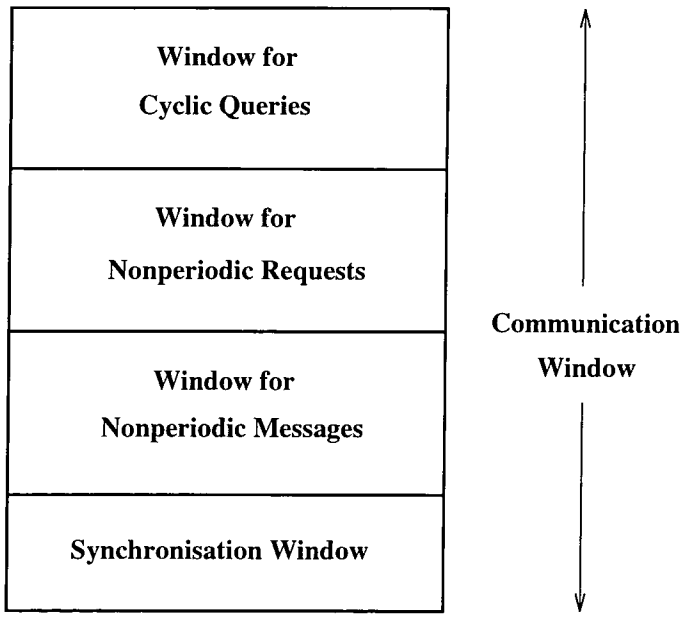


Figure 3.2: The atomic communication interval in FIP is called the Communication Window and it is divided into 4 section for servicing different types of services.

medium algorithm is based on an improved version of the CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) and claims affirm that it is efficient even to heavy networks loads. Due to the access medium algorithm it is not deterministic, and it is not suited for demanding real-time applications.

3.2.7 FIP

FIP is a French proposal for a fieldbus standard based on a shared bus topology which has several international supporters. The system had been designed following a Producer-Distributor-Consumer model which is based on a broadcast mechanism without physical addressing. A single station, called bus arbiter, administrates the shared transmission medium for every producer. Variables are the central communication objects, representing the locations of memory containing a measurement value as well as list of identifiers which request stations through the bus arbitrator or a request for a data transfer. For every variable there can be any number of consumers, but only one producer; therefore they are used instead of the physical address in all the communication transactions. The system, in order to be able to allow real-time applications and deterministic responses, organizes all the com-

munication transactions in communication windows, also called elementary cycles. Each window is divided into four segments each of them devoted to processing a special class of services, namely cyclic and acyclic variables, nonperiodic messages and synchronizing the connected stations (see fig 3.2), under the control of the bus arbitrator. The sequence of of elementary cycles, called macrocycles, which defines the sequence of periodic inquires that the bus arbiter has to send on the network, is determined off-line and cannot be changed at run-time. During a macrocycle a variable can be queried multiple times, giving that variable higher priority. The traffic related with the queries of the variables, associated with the window for cyclic queries, constitutes the larger amount of data traveling on the network. FIP do not allow individual medium access to the interconnected stations, but all the communication are regulated by the bus arbiter. This station broadcasts variable requests, which can be an identifier from the query table or an identifier for a non-periodic service request, to all the connected devices. The node interested in the query (the producer) replies to the bus arbiter by putting the current value of the variable on the bus, while at the same time all the interested nodes (the consumers) read that value from the medium. Finally the bus arbitrator regains control over the medium and can produce the next variable request (see fig. 3.3).

FIP can be implemented on either twisted pair or fiber optics with transmission speeds of 31.25 KBit/s (with cables up to 1500 m), 1 MBit/s and 2.5 MBit/s (with cables up to 500 m). The maximum number of connected stations is 256.
Simulations of the FIP protocol [EK93] demonstrated that the performance is influenced by the number of non-periodic requests and especially from the configuration of the internal resources of the bus arbitrator. In order to get the best results it is necessary to configure for the special requirements both the windows for non-periodic messages and the internal buffers, tasks which request some configuration efforts. However the response times of the periodic traffic are always deterministic and independent from the from the dynamical traffic arrival. It was found that the system is unsuitable for real-time responses within a certain range of requirement profiles.

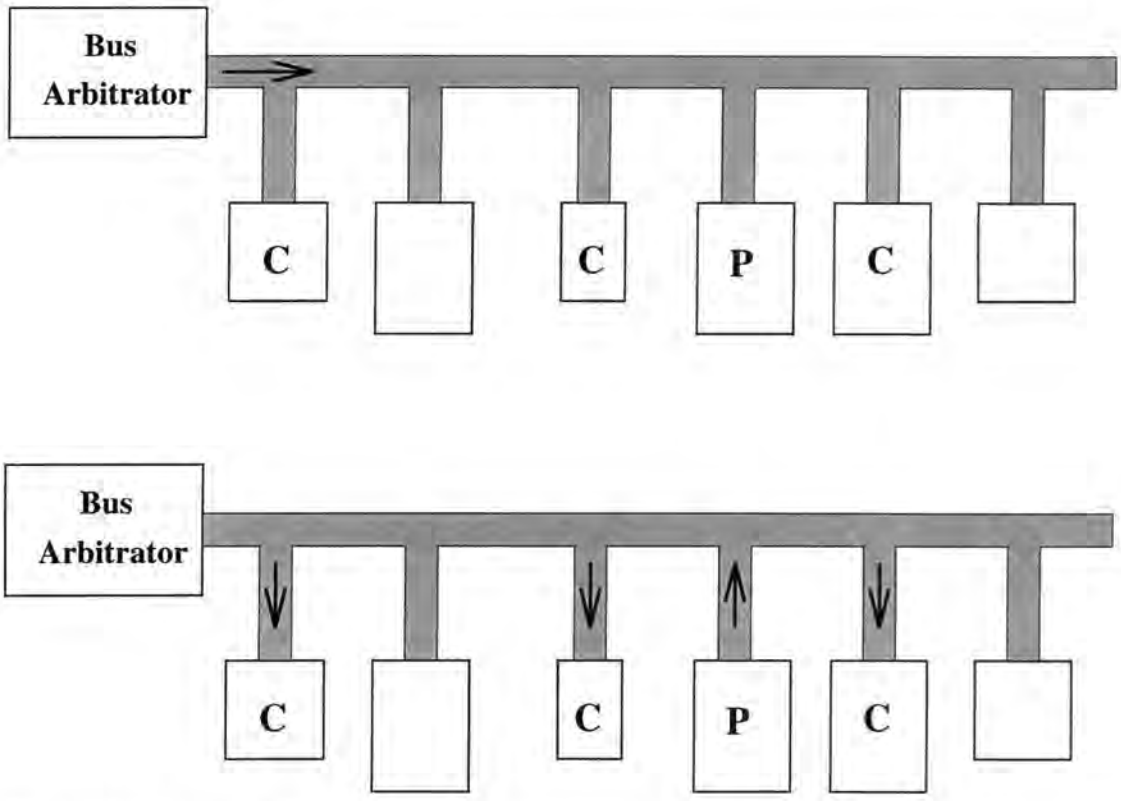


Figure 3.3: The FIP messaging principle. The bus arbitrator sends an identifier and the addressed unit (producer) replies with the actual data which is received by all the consumers (C).

3.2.8 Profibus

Profibus is a high level German fieldbus standard which is suitable for interconnecting both low cost devices, such as sensors and actuators, and more powerful machines such as PLCs and NCs. Its design resembles the model used for the design of miniMAP (a reduced version of the MAP protocol). The network topology is based on the token bus principle (see fig 3.4). Two types of stations are connected along the bus; masters and slaves. When a master receives the token it compares the target rotation time with the actual one and if the second time is inferior it is allowed to retain the token for a certain time and perform its communications activities over the network. Messages are divided into three classes; high priority ones, polling ones (the ones used to interrogate the slaves) and low priority ones. A master holding the token first will transmit the high priority messages followed by the polling ones and then if there is some time left the low priority ones are forwarded to the network. Even if the target rotation time is smaller than the

actual rotation one a master is allowed to send one high priority message. Slaves can access the medium only when they are explicitly commanded by a master.

At the application layer every device is seen as a Virtual Field Device (VFD) which is an interface that gives access to all the objects of a real device that can be communicated with. They are treated as entries in the Object Dictionary, a list which describes all their attributes (see fig 3.5). A revised subset of the Manufacturing Message Specification (MMS), which was designed to be the application layer of MAP, is used to define all the possible services which can be obtained from the network by the interconnected devices.
The maximum, number of devices which can be interconnected is 127. The transmission speed is from 9.6 KBit/s to 500 KBit/s in selectable steps, while the maximum bus length varies 200 m to 1500 m depending on the transmission speed.

In addition to the Profibus FMS, described above, which is suitable for upper level communications, other two variations of the standard have been proposed: Profibus DP for time critical applications with speed up to 12 MBit/s, and Profibus PA for intrinsically safe installations.

Some tests have been performed on the FMS version of Profibus by [EK93]. Simulation results showed that real-time capabilities are guaranteed by selecting the proper selection of the target rotation time. The number of master stations is the critical parameter since too many masters in the logical ring results in poor service when real-time response is required. It has been noted that once a certain amount of high priority messages is exceeded the other types of messages do not have time to be sent along the network (this fact could be critical for slaves that have to send alarms through the network). Lower data rates for Profibus are suitable for automation processes with few transactions and very relaxed real-time requirements. Similarly to FIP, the use of Profibus for real-time communications is not always possible, but it depends on the exact requirement profiles.

However the Profibus organization become aware of the impossibility of using Profibus in all the environments and therefore has proposed other two versions which are suited for more demanding requirements profiles and it is trying to propose Profibus as the *de facto* worldwide fieldbus standard.

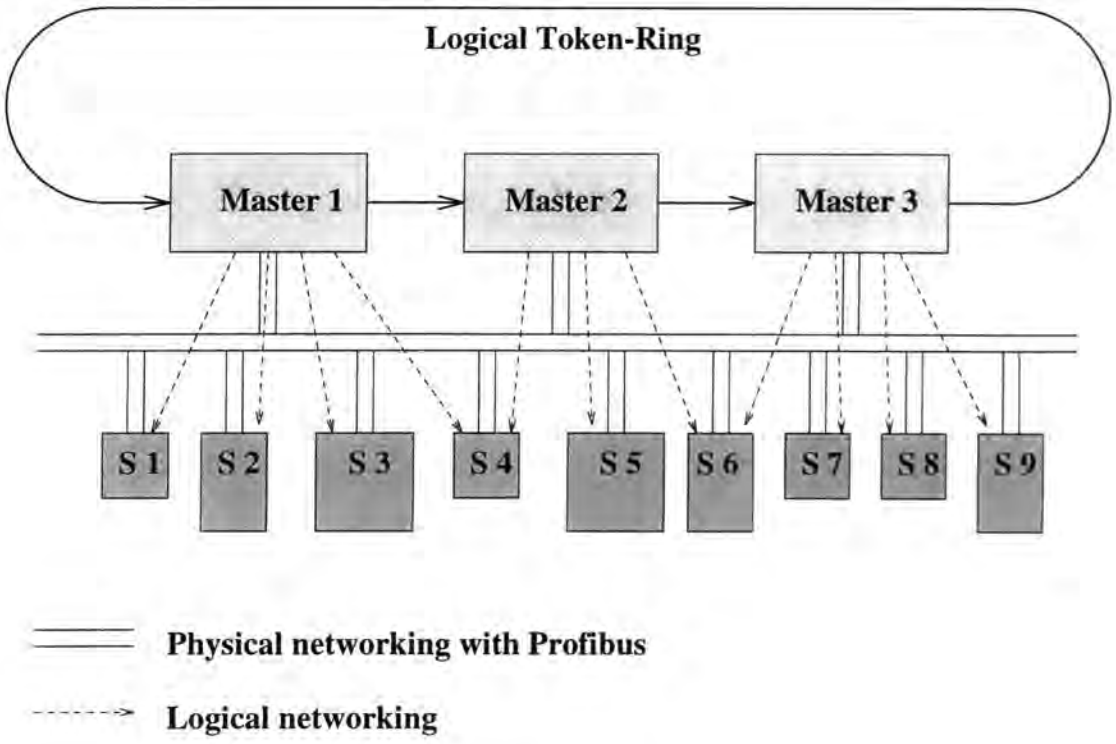


Figure 3.4: The Profibus operating principle. Master stations circulate the token, and when they hold it they can communicate with slave stations.

3.2.9 IEC/ISA SP50

This protocol is currently developed at international level and it is supposed to become the official fieldbus worldwide standard. It has taken advantage of multiple contributions from many other fieldbusses. So far only one part of the protocol, namely the Physical Layer, have been finalized and the other parts are still in the form of drafts. Currently the only medium option is twisted pair, but fiber optic and radio are expected. The bus speed is between 31.25 KBit/s and 2.5 KBit/s (but 10 KBit/s is likely to be introduced). A draft of the Data Link Layer is still under international analysis and it includes several different types of fieldbus access methods, including bus arbitration, central mastership, free and delegated token circulation. The maximum length of the packets is planned to 256 bytes and the maximum number of addressable nodes is supposed to be 256. At the Application Layer level two different system which allow the user to communicate with the system have been proposed, a Device Description Language (DDL) and Functional Blocks. In addition to the seven OSI layer an additional layer called User Layer,

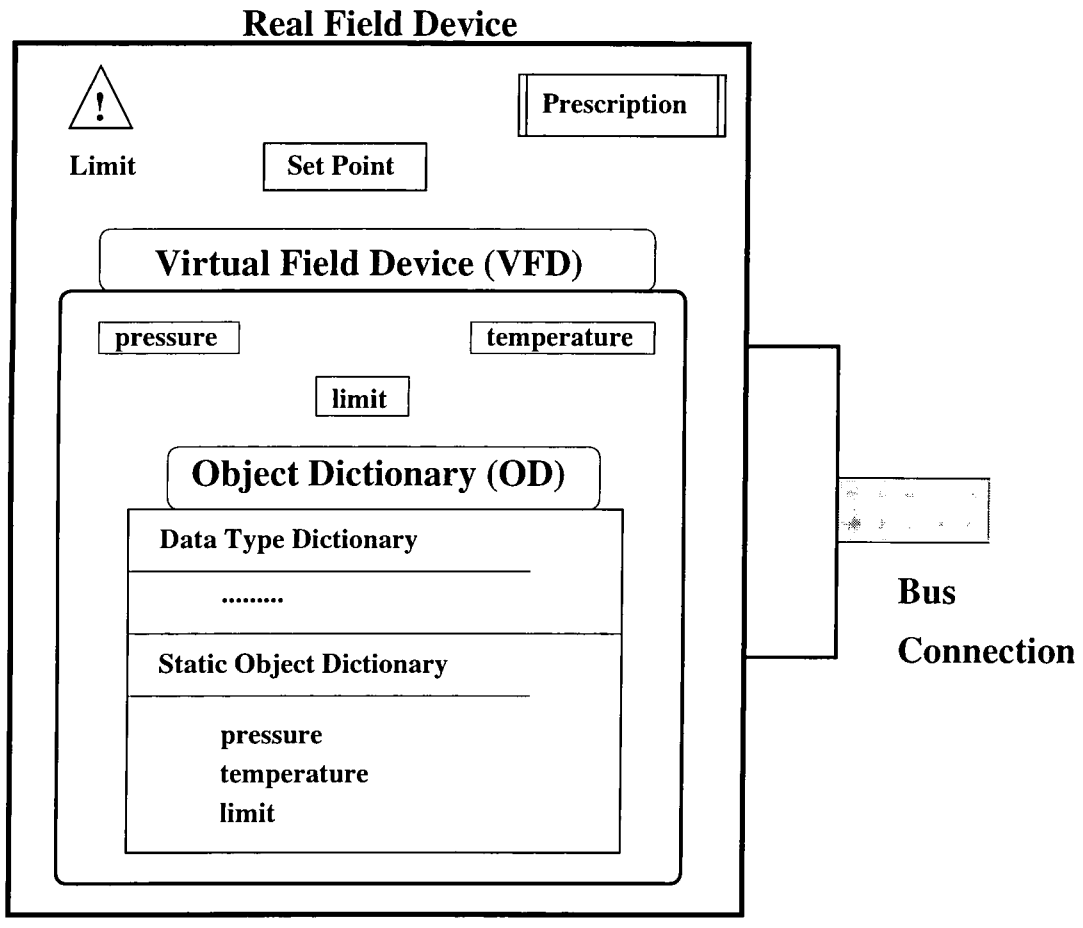


Figure 3.5: Representation of the VFD used in Profibus to offer a standard interface for all the devices connected to the bus.

is under development and it will include all the facilities for high level dialogue between the user and the devices in order to facilitate the system interoperability and integration, and reducing the installation times. The work on the SP50 fieldbus is still far from being completed and it will take time before it will be possible to have products implementing this technology on the market.

3.2.10 Fieldbus Foundation

The Fieldbus Foundation has proposed an international level fieldbus protocol compatible with the IEC/ISA SP50 fieldbus. It has adopted the Physical Layer protocol defined by the IEC/ISA group. A major debate was caused on the definition of the Data Link layer; eventually it was decided to use a combination of a central-

ized master (called active scheduler) and token passing. This solution will allow predictable cyclic updates and cope with asynchronous traffic. The discussion on the upper layers is still open but an approach very similar to the one prospected by the IEC/ISA committee is likely to emerge.

3.3 Ethernet and Internet protocols

The advantages which can be gained from the use of a standard fieldbus system in production or process plants are clear. Reduction in the cost of the required wiring, possibility of remotely reconfiguring a node, and interoperability, which allows the user to choose the device which best suits his application. Operational benefits include higher accuracy, better and safer control, and more information. The introduction of low-cost computing power in each field device will allow the replacement of centralized control systems with distributed control networks improving performance and reducing the problems related with system. In addition maintenance will benefit from increased reliability, more diagnostics and possibility of using devices produced by different suppliers [Wat94].

However there are several problems related to the actual state of the fieldbus technology which may explain why the users may not want to adopt any of the proposed solutions. First of all there are too many solutions proposed and it is likely that some of them will be abandoned or will suffer of obsolescence in the near future. Other more important issues are raised by D. Roeder [Roe95] who points out that even if the difference in price between normal and smart equipment is eroding, it is essential for the designers to have simple rules for network design and a suitable user interface; installation and commissioning must take advantage of the simple and proven technology already available. At the present there is not enough knowledge about which are the best options for fieldbus optimum control, safety, reliability, operability, and damage limitation. Only tests on pilot plants will be able to give an answer to these questions. Taking into account all these issues it is easy to see why several companies do not want to risk full scale implementations which may involve large cost and losses for them.

At the present due to the number of different network solutions available it looks as if the major activity will be in the implementation of gateways between proprietary products and open architectures, since it is much easier to make a whole set of devices work with any bus through a gateway than to convert the entire product line to operate with a different bus, even if this means a decrease in the performance of the system whenever a communication through a gateway is needed [Pin95].

Fieldbus protocols have many common features, and yet are generally incompatible with each other. In addition they are subject to a rapid evolution due to the changes of the requests of the market. These facts have led to a reconsideration of the use of Ethernet and the Internet Protocols for implementing industrial communications protocols. Ethernet is a well established technology developed in the '70s, used for connecting computers together in local area networks. Ethernet networks are now commonly used as the basis for data communication networks in industry. Many custom applications have been developed for industrial environments. In addition Internet Protocols are the standard way of communication of all the computers connected to the Internet. These protocols are easy to use, are fully defined and represent an accepted world-wide data communication protocol.

3.3.1 TCP/IP suite

In any overall digital transmission protocol, the definition of the protocol stack is crucial in enabling successful transmission to be achieved. The Ethernet layer forms the lowest element of the protocol stack and it is concerned with the transmission of units of data (called frames) between machines on the same physical network. It defines the details of the types of cable that can be used, the hardware interface, the rules for accessing the transmission medium, and the format of the messages. The part which is involved in processing the incoming messages and sending the outgoing ones is always implemented in silicon. The chips are manufactured and sourced by different vendors, and are generally available at low cost. Ethernet

is usually used in conjunction with the TCP/IP set of protocols (also known as Interned protocols). The Internet Protocol (IP) implements the lower layer services in the TCP/IP stack. It is implemented directly on top of the Ethernet protocol. It is concerned with transporting packets of data from one machine to the other, but it adds the important capability of facilitating the routing through intermediate gateway machines, in order to forward messages directed to machines which are not directly connected on the same physical network. The next layer in the protocol stack is the transport layer. Climbing up the stack of the TCP/IP suite, there are two transport protocols. These are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). They have different features which will be highlighted in the next subsections. On top of them protocols for carrying out specific tasks, such as transferring files between computers, sending mail, or finding out who is logged in on another computer are implemented.

User Datagram Protocol (UDP)

The UDP [Pos80] protocol provides a simple transmission protocol which is able to distinguish among multiple destinations on a single machine. In addition to the application data, the UDP protocol tags the message with both a 16-bit destination port number and a source port number, allowing the UDP software to deliver the message to the correct recipient, and for the recipient to send a reply. UDP provides an "unreliable connectionless" message delivery service. This means that it does not have any acknowledgment mechanism which assures that messages arrive at their destination. Moreover, it does not sort the messages if they reach the receiver out of sequence, and there is no feedback mechanism to control the rate at which the data flows between the machines. Thus, messages can be lost, duplicated, arrive out of order or arrive faster than the recipient can process them. However, UDP adds a 16 bit checksum to the message which allows the receiving UDP layer to check the validity of the incoming data. UDP does not fragment data, and therefore the application programs have the responsibility for dividing the message into suitably sized chunks (generally the maximum length is 64 Kbytes), as well as implementing their retransmission, reassembly, flow control, and congestion avoidance routines if

any are required.

Transmission Control Protocol (TCP)

The TCP protocol [Pos81] is the reliable equivalent of UDP. It includes a method for delivering messages to specific recipients within a machine, based on a 16-bit port number (however the TCP and UDP spaces are separated). TCP is a connection oriented protocol, which means that before transmitting, both ends of the connection must agree that the communication is desired. It provides a corruption free, in-sequence, and reliable delivery service between transport end points, thanks to built in error checking and retransmission procedures. There is no upper limit to the dimensions of the messages which can be sent, as TCP automatically fragments the data into packets of suitable size. Moreover, in order to resolve efficient transmission and flow control, it implements an algorithm, called "sliding window", which makes it possible to send multiple packets before an acknowledgment arrives. As TCP assumes little about the underlying communication systems, it can be used with a large variety of delivery systems.

3.3.2 Ethernet in industrial applications

Ethernet is already used for industrial control applications, generally for data recording and management information tasks. This shows that even if it has been designed for interconnecting personal computers in an office environment, it can be successfully used in more demanding environments.

The fact that Ethernet was the first LAN technology endorsed by many vendors is often used for explaining its success on plant floor. This is a very important point, but the real cause of the success of Ethernet has to be found in its features and especially in its intrinsic reliability, which is a key factor in all industrial applications.

First of all the nominal transmission speed of Ethernet is 10 Mbit/s, which is far above the maximum transmission speed of any other fieldbus available at the present on the market. In addition the standard defines a wide range of transmis-

sion media; coaxial cables, baseband and broadband versions (the latter is not usually used due to its cost), twisted-pair, and fiber-optic. In addition several vendors offer versions of wireless Ethernet using lasers, microwaves, and spread-spectrum radio transmissions. However, none of the wireless Ethernet are organized by any standard body, and therefore interconnecting equipment from different manufacturers could generate problems. Among the other fieldbusses available on the market only LonWorks, produced by Echelon, has a larger spectrum of media supported, while most others tend to employ twisted pair interconnection. The possibility of corruption of packets travelling on the network due to electromagnetic interference from industrial machines does not exist since coaxial cables defined in the standard have higher levels of immunity [Ada90], and category 5 UTP (twisted pair) has been installed in a variety of manufacturing plants without any problem. This allows the use of Ethernet in plant-floor areas where electromagnetic fields radiate from production devices. However in order to eliminate completely the problem of electomagnetic interference, optical-fibre, which is completely immune to electromagnetic noise, can be employed at an increased cost of cabling. Cables can span factory-wide distances if appropriate devices are used. The possibility to choose various types of cabling allows the designer to select the solution which best suits the plant. Maintenance, modification and expansion of an Ethernet installation are not problematic since stations are easily added or deleted simply by activating or deactivating them, without the need of shutting down the system as in token approaches. The failure of a station does not affect the behaviour of the network.

In proposing Ethernet for time critical control purposes, the major concern is that as Ethernet is not a deterministic network, the time of transmission cannot be guaranteed, and its inherent property of collision detection and retry in the transmission process compromises its real-time capability. It is important to study the actual times in question and see to what extent the system can be used in time-critical applications. Results of such a study implemented by the author are presented below.

3.4 Results of timing experiments

The performance of the TCP and UDP protocols of two different UNIX implementations on an Ethernet network was investigated by measuring the time delay as a function of the message size. The test of the two protocols was performed using two pairs of identical machines in order to compare the performance of two operating systems independently from the hardware.

The computers were a pair of Intel 80486 33 MHz with 8 Mbytes of RAM, 1.2 Gbytes IDE HD using WD8003 Ethernet cards. They were interconnected on a 10 MBit/s isolated Ethernet, thus eliminating the possibility of collisions of packets. Two operating system were installed and tested; BSDI BSD/OS 2.1 and Linux 2.0.29. BSD/OS is a commercial implementation developed by Berkeley Software Design Inc of the shareware distribution of the BSD UNIX operating system. Linux is a freely available UNIX version, distributed under the GNU General Public License, whose development is the result of the effort of hundreds of people around the world. Both of them can be used on several different platforms including Intel 80386, or greater, and compatible processors.

The test permitted the comparison of the performance of a shareware implementation of UNIX with a semi-commercial one, to assess whether freely available software can compete in performance and reliability with the commercial products.

For every message length, 100 trials were performed in sets of 50. The results give an account of how the protocols perform in an ideal scenario when the machines are connected to an idle network. The objective was to make a fair comparison of the two UNIX implementations by eliminating the likelihood of degradation of the transmission times due to packet collisions along the Ethernet. It was decided to use the default values associated with the protocols using the default settings of the installations. By not altering the `TCP_NODELAY` socket option, which indicates whether the outgoing data is pushed immediately to the network interface instead of being buffered and processed in larger quantities, the Nagle tinygram avoidance algorithm [Nag84] was *not* set during the experiments. In order to avoid difficulties of clock synchronization between the different machines used in the experiments,

all the measurements were made by taking the "round trip time" (double transit time) of the data from host A to host B and back again.

The timing procedures for both TCP and UDP protocols are very similar, due to the intrinsic resemblance of the two protocols. The software used for taking the measurements consists of two programs; the first one runs on the server machine and the second one on the client. The software on the server creates a socket and binds it to a pre-defined port number and waits. The client software, for each message sent, creates a socket, and binds it to the same port number of the server. It then sends a short message informing the server of the amount of data that it is going to transmit and waits an acknowledgment; this operation is necessary in order to allow the server to prepare an incoming buffer of a suitable size. Then the timer starts, and the client sends the message to the server, which then sends it back. When the whole message had been received by the client, the timer is halted and the delay is calculated. Subsequently, the buffers are freed and the sockets closed. The cycle repeats itself again for the next transmission.

Similar tests have been carried out on pairs of platform running SunOS 5.5.1 (Sparc Ultra 1/140, 64 Mbytes RAM, 2 Gbytes HD and custom Ethernet card) and HP.UX A.09.05 (Hewlett Packard Apollo series 700, HP 710, with 16 Mbytes RAM and custom Ethernet card). SunOS (Solaris) is a UNIX implementation developed by Sun Microsystems for their own computers. HP-UX has been developed by Hewlett Packard for its own line of workstations. Unfortunately it was not possible to perform all the experiments using an isolated network. Therefore in order to minimize the effect of the local traffic the tests were executed during the night at times when the overall network usage was very low, on unloaded systems (the only user's processes active on the platform were the ones involved with the test program). For every message length, 100 trials were performed in sets of 25 at different times during the night to reduce the effects of temporary traffic on the network. The results attempt to give an account of how the protocols can be expected to perform on an isolated network and therefore can be compared with the other data gathered from the Intel platforms.

3.4.1 Discussion of the results

In figure 3.6, and 3.7, round trip times for the messages varying in size between 100 bytes and 900 Kbytes, using the UDP, and TCP protocols, implemented on the UNIX systems, are shown on scatter plots.

First of all it is clear that the performance of the transmission protocol is influenced by the UNIX implementation used. It can be seen that the time required for transmitting a packet may not follow a logical pattern, and may also vary significantly depending of the size of the message. Moreover the behaviour of the system does not follow a smooth curve and sometimes the time required for a round trip deviates considerably from the expected value. Direct comparison of the performance of the operating system is possible only in the case of BSD/OS and Linux, while the other tests provide other information on the system behaviour since the platforms and the test conditions are different.

The plots of the round trip times versus the message length show that the distribution of the data is quite often skewed. This has important implications for the choice of statistical measures which are suitable for characterizing the set. A common form of representing such information is by the use of the *mean* of the data values [BMK88, BBV91, HS89]. For systems which interact with the external world, not only with real-time activities, but also for example with NFS or WWW servers, this can give misleading indications of the likely performance. Instead it is suggested that the *median* is better guide to the typical performance of the systems, particularly when the likelihood of collisions on the network is taken into account. The use of the median tends to eliminate the influence of occasional collisions and other spurious events recorded during the test measurements.

Other published approaches define the throughput of a system by a single number [Svo90, LB96]. This is misleading, as the throughput of the system is dependent on the length of the messages sent. This fact is illustrated in the case of the Solaris operating system in figure 3.8 where the throughput steadily increases up to messages of 2000 bytes, and after that it reaches a plateau. This is due to the buffering operations which achieve increased efficiency with large messages. Similar graphs produced for Linux 1.2.13 showed that the throughput increased up to 50 Kbytes

before reaching a constant value.

Lastly it is common in the literature reporting these types of measurements to use the variance, or alternatively the standard deviation, as index of scatter of the data. This measure is best suited to unimodal symmetric sets, and since the data in most of the cases has a non-symmetric distributions, *percentiles* might be more meaningful.

In figure 3.6 the performance of the UDP protocol on the four pairs of platforms is shown. The length of the longest message was 9000 bytes due to the limitations of the buffer used by the `send()` function in conjunction with this protocol. It has been observed that the performance of the protocol shows a smooth and consistent behaviour with all the UNIX implementations, and only some scattered points are present. In comparison with the other protocols tested, UDP achieves the shortest round trips due to its simplicity. The smaller round trip times are associated with the tests performed on the Sparc Ultra. This result does not surprise since it is a very powerful platform in comparison with the others. It is possible to notice that Linux and BSD/OS have comparable round trip times for smaller packets, but Linux performs slightly better for larger packets. Experiments performed with the BSD/OS 2.0 on an Intel 486 66 MHz and SunOS 4.1.3 and SunOS 5.5.1 on a Sparc 4 showed more scattered points and irregularities (the relevant graphs are presented in appendix A).

In figure 3.7 the performance of the four implementations of the TCP protocol are shown. These graphs display irregularities and scatter values. Some of the scatter present in the bottom graphs might be produced by collisions of packets on the network, but as the experiments were performed in the middle of the night, when the traffic was low, it is more likely that it is due to kernel activities. Among these the most likely to occur are associated with page optimization techniques which avoid copying complete data pages by mapping the pages from the input buffer into the kernel. The performance of the memory allocator may also play an important role since irregularities start to appear after 8000 bytes when the dimension of the packets exceeded the dimension of the input and output buffers and new space needs to be allocated in order to accommodate them. In addition

the limited Ethernet card buffer size may be the cause of some of the irregularities present in all the graphs.

In the results for the BSD/OS, it is possible to see that between 3000 bytes and 20 Kbytes the transmission time of the messages is size independent and larger than expected. For larger messages the behaviour is even more complex as there are several discrete values which the transfer time can assume independently of the message size. Several experiments have been carried out in order to try to locate the source of this phenomenum. It was recorded on the BSD/OS 2.0 version as well (see appendix A). Replacing the WD8003, 8 bits Ethernet cards, with NE2000, 16 bits ones, did not modify the results, showing that the irregularities are not associated with the Ethernet card adopted. Computers have been swapped, and replaced, but consistent behaviour has been observed. A second run of tests of the TCP protocol with the Nagle tinygram avoidance algorithm disabled was carried out, but the results showed the same pattern of irregularities present in the graph reported in figure 3.7. It was discovered that the problem starts to manifest itself when the dimensions of the packet are 2048, which is likely to indicate that one of the buffers used by the system for handling TCP messages becomes full and new space needs to be allocated. Using a digital oscilloscope it was possible to get traces of the traffic of the Ethernet packets on the network. This, in conjunction with the data of the timers used to calculate the time required by each operation, makes it possible to draw some conclusions. All further tests were carried out transmitting packets of 4000 bytes. It is possible to notice that the `send()` function returns before the packets are physically sent on the network. This means that the TCP code of BSD/OS just passes the packet to the underlying layer and returns without waiting for the actual transmission of the packet. The time between the instant when the `send()` returns and the actual transmission of the packet is quite small (less than 10 milliseconds), but occasionally it may be much longer (more then 100 milliseconds). This applies on both the client and server sides. In some cases the expected transmission time has been recorded (about 3 cases in 400 transmissions), but sometimes (about 10 times in 400 transmissions) the round-trip time was greater than 0.3 seconds. Due to the complexity of the system it is not possi-

ble to identify precisely the cause of the problem. BSD/OS is a complex operating system made by a large number of modules, whose interactions are sometimes not deterministic. BSD/OS support has been contacted, but no one has so far been able to provide an explanation. Some of the traces recorded by the oscilloscope are reported in appendix A.

The graph associated with the Linux performance shows a consistent behaviour with the exception of messages whose length is 2000, 7000 and 8000 bytes. When the test were performed again with the Nagle tinygram avoidance algorithm disabled both the irregularities disappeared; a fact which suggests that they are associated with internal TCP buffers timeouts. This behaviour has not been seen when Linux 1.2.13 was tested on the same pair of platforms. It is possible to notice that for small packets the performance of Linux and BSD/OS are comparable, while for packets larger than 2000 bytes Linux outperforms BSD/OS.

HP-UX shows a performance with some irregularities. Similar irregularities have been reported in [BBV91], using the XNS protocol, and in [HS89], with OSI communications protocols, and here the cause was identified as related to fragmentation of the messages. It is likely that fragmentation is one of the causes, but it is suggested that the most important factor is associated with the buffering algorithm associated with the output side of the TCP implementation. This suggestion is supported by results published in [PVRS95] where measurements show that the "CPU demand per message" for sending a message between two HP700 (HP-UX A.08.07) using TCP had a large variability which was most noticeable on the sender side. In addition internal timeouts of the TCP software may play an important role in explaining some of the irregularities. The irregularity associated with packets whose length is 9000 bytes disappears when the Nagle tinygram avoidance algorithm is disabled, which indicates that the problem is associated with timeouts when new buffers are created and new data is awaited before sending the message.

Solaris shows a graph which is almost always consistent with very few irregularities. This is surprising since similar tests carried with the same OS on a Sparc 4 produced a graph with many scattered values (see appendix A). The scattering of the values recorded on the Sparc 4 may be associated with kernel activity during

transfer operations. Due to the speed of the processor of the Sparc Ultra, the kernel activity does not interfere with the transmission operations producing a smooth graph.

Unfortunately, direct comparison of the data presented here with results reported in other work elsewhere is not always possible, as the software used to carry out the tests and the way in which the tests are performed is usually author specific. However there are some publications which may be used as a base for comparison. The results presented in [JP94] indicate that the performance of a Sun Microsystems IPC workstation running SunOS 4.1.2 is comparable to the same machine running Solaris reported in our research, and the work reported in [BBV91] shows that the performance of the TCP protocol under MS-DOS (the processor is not specified) is worse than that presented here, and the maximum throughput is reached at messages of 3000 bytes.

Before starting to discuss the preceding results and the possibility of using Ethernet in conjunction with Internet protocols in industrial applications, it is necessary to point out that real-time applications are divided into *hard*, where a missed deadline produces an unrecoverable or mission critical error, and *soft*, where missed deadlines cause only a non-critical degradation of the performance. In addition real-time systems have timescales which can vary greatly (from hours to milliseconds). A table summarizing the requirements of the messages transmitted in an industrial environment is presented in table 3.2 [Ben93]:

The range of the requirements of real-time systems is so wide that in some cases the lack of determinism of Ethernet may not be a problem.

Several industrial installation studies [BBV91, BMK88] showed that Ethernet performs well even on medium load. From these studies it is possible to see that a load between 10% to 20% ensures a behaviour which is close to the deterministic. In addition it has been shown that the peak load of manufacturing sites network utilization rarely is great than 15% while the load average is usually around 1% [Ada88]. This is especially true in all the environments where messages are relatively small and far apart (tens of milliseconds), where the greatest part of the

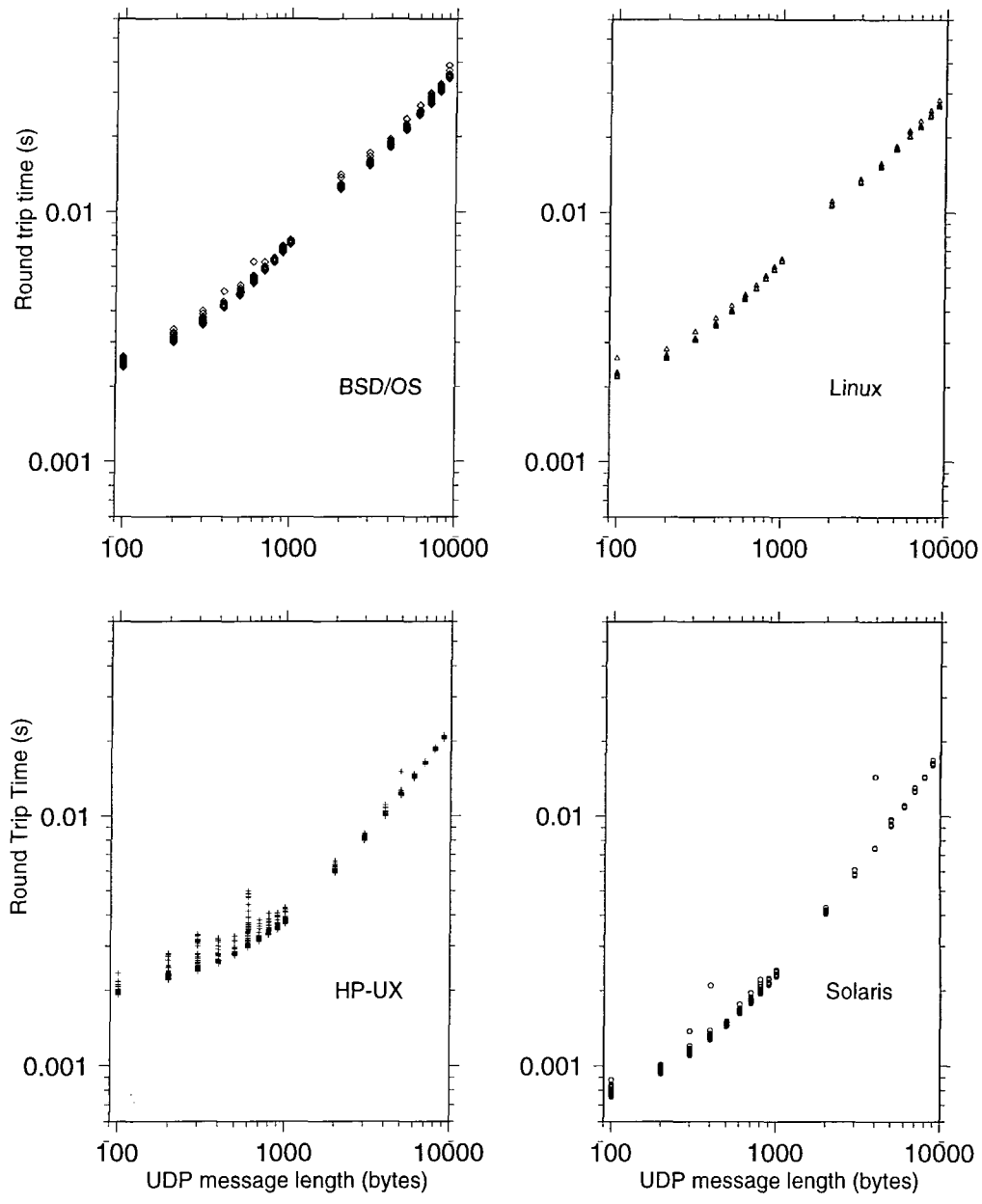


Figure 3.6: UDP transmission protocol performance on four different UNIX implementations

indeterminism is due to content switch effects in kernels, delays in application programs, protocol queues and overheads, and hardware slowness. A work from US national Bureau of Standards reached the conclusion that "Ethernet is a reasonable access method for time-critical applications on small plant LANs if loads of less than 40% are anticipated" [WH86]. 20% load on an Ethernet network is larger than the maximum bandwidth that any other fieldbus can offer on comparable distances. Moreover in recent times industrial devices such PLC are intelligent enough

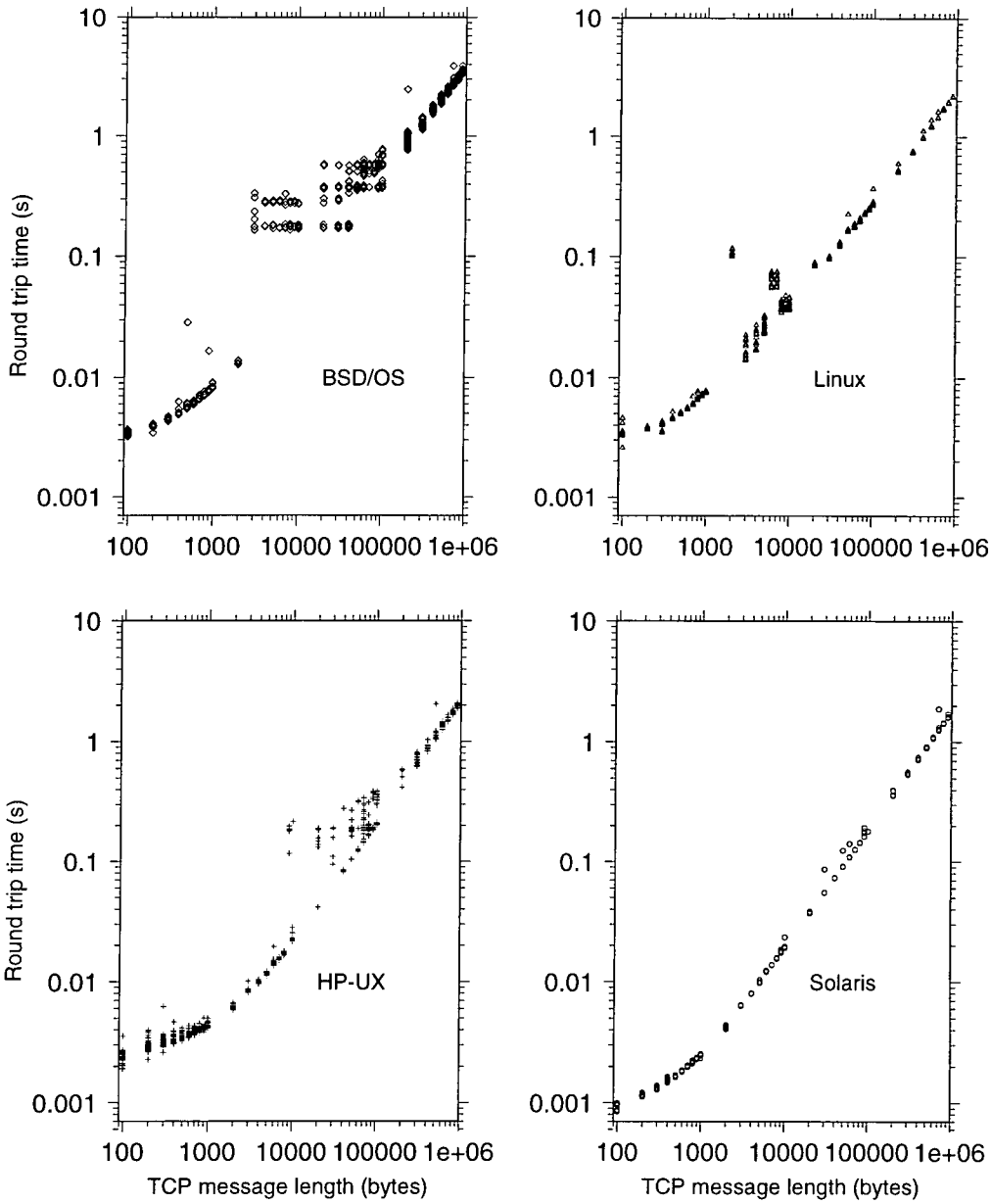


Figure 3.7: TCP transmission protocol performance on four different UNIX implementations

to work without having a maximum response time, and therefore some delays may be acceptable.

From the graphs presented here it is possible to see that the the overhead of processing a TCP message is a little higher than a UDP packet, but the difference is not great. The additional cost of maintaining timers and windows sizes of the TCP protocol are not relevant on fast CPUs. Since the time spent by packets travelling along the cable can be measured in microseconds, the greatest portion of the transmission time is spent at both ends of the process, when the packet is sent

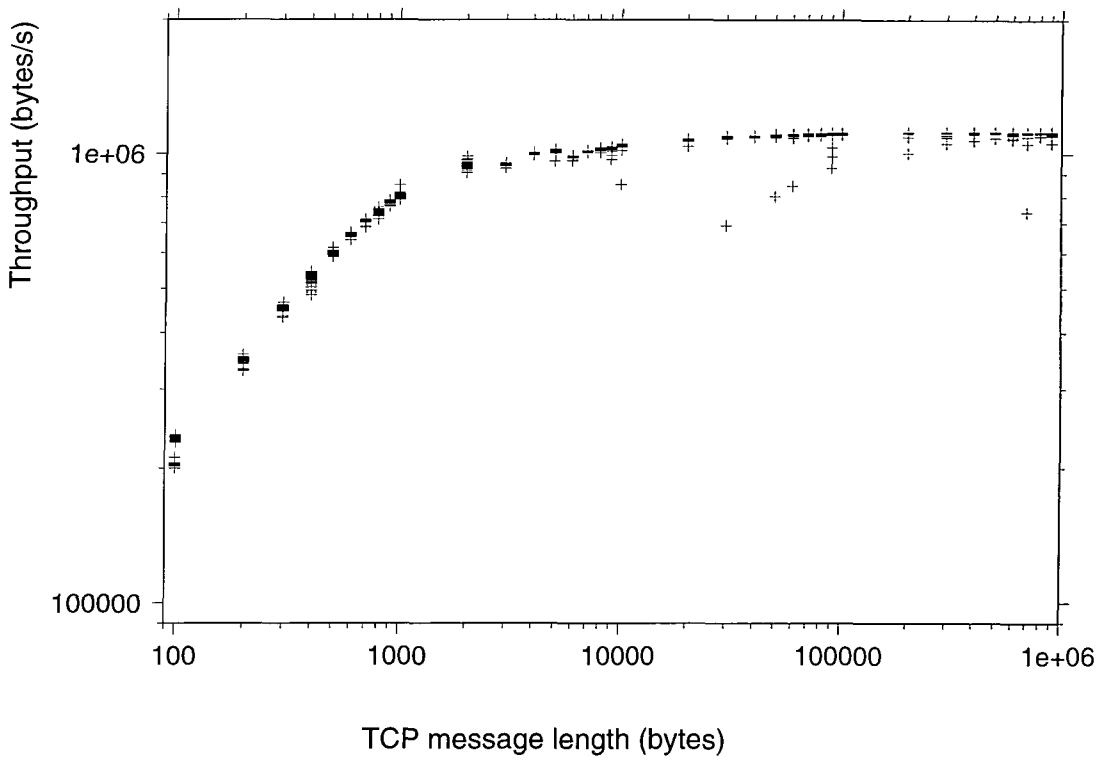


Figure 3.8: Throughput of the Solaris operating system as a function of the message size on a Sparc Ultra

	graphic files	data files	numeric control programs	syncr signals	nominal and actual value signals	event messages
allowed delay	1-100 s	1-100 s	1-100 s	1-100 ms	20-100 ms	0.1-80 ms
message length	>10 Kbits	1-10 Kbits	>10 Kbits	8-64 bits	<10 Kbits	8-64 bits
frequency of appearence	rarely	very rarely	very rarely	very frequently	frequently	rarely
classification	non time-critical messages			time-critical messages		

Table 3.2: Type of messages and the related technical data characteristics

and received. The time spent in processing operations depends on the speed of the CPU, and slower machines require more time to process incoming and outgoing messages [BP96]. A series of graphs of the test results reporting comparison of the performance of Linux 1.2.13 on three different pairs of identical platforms is reported in appendix A.

Performance is not the most important issue over the choice of protocol, which should be determined by the application service. If the messages travelling along the network do not need to be acknowledged (for example in data collection applications) and the loss of a packet can be accepted, UDP is the best choice, while if it is necessary to implement a system which is connection oriented, where all the packets need to be acknowledged, TCP is the best choice. The fact that UDP is defined an insecure protocol does not mean that it is unreliable, since losses of packets may happen only on long distances between computers hundreds of kilometers apart, when a packet has to go through several gateways. It is important to remember that NFS runs happily using UDP packets in campus-wide LANs. If losses of packets are reported in a LAN environment this usually means that there are problems in the cabling of the system or in some of its components. The main problem with TCP is that it needs a lot of memory space in comparison with UDP, and this fact is extremely important in all small embedded application where extra memory might be an expensive option. A stripped down version of UDP could need as much as 200 lines of code to be implemented [Alh96]. In addition TCP makes more intensive demands on the system's network buffering since TCP cannot discard data it has sent until it has been acknowledged by the TCP at the other end. It may be also difficult to get real time responses with TCP if some package loss happens and retransmission is needed, while with UDP it is always possible to have a more accurate control since no retransmissions are possible. Using UDP it is not possible to know whether the receiving station is alive or not, since no acknowledge is requested, while TCP attempts the connection for a time up to 2 minutes before declaring the host unreachable (TCP was designed for global networks). This inconvenience can be solved if appropriate application level watchdogs are implemented.

The measurements presented here represent the behaviour of the systems transmitting on an idle or lightly loaded network. In a real world situation other factors further reduce system performance. Scheduling interaction between the processes which make use of the network and the other tasks running on the same machine will occur. ARP queries, when made, may be also a time consuming activity. In

addition there is the possibility of collision of packets along the network, and if packets greater then 20 Kbytes are transmitted, the Ethernet capture effect, which denies access to the network to some stations, may start to affect the performance. Lastly there are I/O and bus interactions on the card when incoming and outgoing messages are handled by the system at the same time.

However the measurements presented here show the range of timings and likely variations for the unloaded network. They are indicative of the round trip times and as such are of importance in finding system performance for time critical applications.

Costs are a very important factor in the selection of a product. At the present the main focus of the customers is initial costs, which include the purchasing of hardware and software, the design, installation and startup. The suppliers do not stress the fact that the ongoing costs, which include maintaining the hardware and the software, people to operate the network and expansion and configuration changes, are, in most of the cases, the most onerous for the installation. An application using Ethernet and Internet Protocols may have higher initial costs, but in the long run it is likely to be less expensive. While fieldbusses are new technologies Ethernet and the Internet protocols are well established products with an immense amount of development behind them. The know-how is already available and does not need to be developed from scratch, as in the case of fieldbusses. Ethernet chips are available to everybody and there is no need to become a member of an association in order to be able to acquire and sell products using this technology (however there is a central authority which coordinates the assignment of the Ethernet numbers). Many complementary protocols have been developed in order to standardize the approach for solving some of the more common problems. There are several tools already available and additional protocols have been defined to solve some of the more common needs. For example it is possible for a diskless station to boot from the network (see RFC 951 [CG85]) and to dynamically configure a new host on the network (see RFC 1531 [Dro93]). This second protocol is extremely important in all the applications where there is a number of nodes to configure as it carries out the task automatically.

This discussion does not aim to prove that Ethernet is suitable for all types of applications. A careful analysis of the requirements of the system is necessary before deciding which is the best solution to apply. If Ethernet is chosen, particular attention has to be placed in the selection of the Internet Protocol implementation since different implementations, and especially the ones designed for embedded systems, have slightly different characteristics which can make the difference when used in the field. From the graphs presented here it is possible to see that the transmission times for packets sizes used in a control environment are always in the order of milliseconds. A collision may delay a packet by up to 50 ms. Since in low traffic conditions (load less than 5%) collisions are extremely rare it is safe to assume that the largest majority of the packets will reach the destination in time and only few of them will suffer delays greater than 50 ms. Therefore Ethernet and the Internet protocols can be a valid option in all the applications which can satisfy these time constraints. For busier networks the limit can be increased to 100 ms.

Ethernet and the Internet Protocols provide only the lower levels of a protocol stack implementation, but the same is true for CAN, which defined only the first two layers of the OSI stack, and this protocol has been extensively used in many hard real-time applications. This simply means that the higher levels of the application have to be custom developed (this may not be a problem in some industrial application where the scale-economic factors are not applicable). However there are already examples of developing standards, such as MITS [For96].

3.5 Conclusion

In this chapter a review of the recently available fieldbusses has been presented and their comparison with communication systems based on Ethernet has been given.

Fieldbusses can provide several advantages when used in production or process plants. However their number suggests that some of them will be abandoned in future, and in particular at present there is not much knowledge in the field. These facts are slowing down their adoption in manufacturing plants. Since embedding

PCs inside manufacturing devices is becoming more common, this suggests that Ethernet, a well proved, inexpensive, standard technology, can become a valuable alternative to the various fieldbusses for providing a communication means for industrial machinery.

The results of the performance of Ethernet used in conjunction with several popular network transmission protocols on different UNIX implementation has been reported. From these tests it can be seen that the time required to transmit a packet varies significantly depending on the size of the message, and the relationship is not a simple function, but it is heavily dependent on the implementation.

The use of pairs of identical platforms enabled the comparison of the operating system to be made in the same condition, and the tests on the other platforms provided data which allows evaluation of the importance of the computer architecture on the performance. In addition the platforms used are amongst most popular among the UNIX users and provide interesting results for a large community of users. The study has revealed that one of the operating systems has major problems related with the implementation of the networking software. The UDP protocol is the one which presents the lowest round trip times, and gives more consistent results with only minor scattering problems. TCP may reveal inconsistent results as soon as the length of the messages is greater than the dimension of the input and output buffers. Therefore an application program using these protocols should use messages which are located in an area where a predictable behaviour is observed. In the case of the Linux and Solaris operating systems, in contrast, the best approach would consist in sending longer messages, balancing the improvement in speed offered by the greater throughput against the possible disadvantage of the initiation of the Ethernet capture effect. If especially high performance is needed, such as in hard real-time systems, setting the `TCP_NODELAY` option on the sockets, in order to disable the Nagle algorithm and allow the fast delivery of small packets (with an increase in the traffic along the network) and the use of very high performance cards which implement capture effect avoidance (e.g. the AMD PCNet/ISA PCNet/VLB PCNet/PCI or the SMC Etherpower and other Tulip based boards) may be good solutions.

The network implementation of Linux, which is a public domain operating system, shows a consistent and logical behaviour. This implies that shareware software can compete in quality and performance with commercial products, and this makes it a candidate operating system for implementing high performance distributed applications.

High speed modern computer platforms provide transmission times using Ethernet and Internet protocols which can be successfully employed in what have traditionally been considered real-time applications. The robustness of the transmission media, and the wide availability of hardware and software for UNIX type implementations, offer a well documented and tested solution to the implementation of industrial networks for applications of real-time control.

Chapter 4

Automation in the footwear industry

4.1 Introduction

The footwear industry offers examples of manufacturing processes which require the production of small batches of shoe components which must be made with precision and in short times. Until recently most of the production processes were manual, or involved machines used as stand-alone devices requiring human operators. However the introduction of automation is changing the industry and at present there are examples of industrial machines which can carry out certain processing and assembly operations in a completely automatic fashion. The goal of designing an automatic system for the complete assembly of a shoe is still remote, but at present it is possible to identify some cell groupings where linked automation is feasible.

Shoemaking is a process comprising several operations which transform the raw materials in the finished product. The operations involved in the assembly of the shoe can be grouped in the following sections:

- product development
- cutting;

- closing;
- making;
- finishing.

During the time spent at the BU in Leicester, and thanks to several visits to shoe factories, it has been possible to analyze all the processes involved in shoemaking, and isolate the areas where a communication link among several machines would be useful in order to improve the production flow.

In the section 4.2 an overview of the operations performed in the different production areas is given. A full description of the operations involved in shoemaking and other activities relevant for the footwear industry is given in [Mil78]. The use of communication protocols at present in the footwear industry is presented in section 4.3, while section 4.4 presents an overview on the future developments in the area. Section 4.5 gives an overview on the devices which could be included in a manufacturing cells for the footwear industry. Finally two examples of networked workcells are presented in section 4.6.

4.2 Shoemaking processes

4.2.1 Product development

Product development is the process of designing new styles or adapting existing ones and then specifying the materials and the components. The designer traces the shoe using drawing tables or in more advanced industries with the help of CAD systems specially designed for shoe-designing. The project defines the materials, the components and their shapes. The data about the shape of the upper components of the shoe is essential to produce knives (called press knives) used to cut the pieces of the upper part of the shoe. Once they are ready the industrial production of a style of shoe can start.

4.2.2 Cutting

In the cutting section the different parts for the upper of the boot or shoe are cut from leather or other materials. The work performed in the *clicking room* (the name given to area of the factory where the cutting operation is performed) is very important because of the need for accuracy in cutting the shoe's components, and reducing the waste material. Poor cutting can affect a great number of subsequent operations and the profitability of the company.

The materials used for the production of the shoe uppers can be divided into leather and synthetic materials.

The reasons why leather is considered a suitable material for shoe uppers are various. It is an elastic material able to recover its original shape without damage (for example when a shoe is flexed during the walking activity). It is also a plastic material which allows the shoemakers to mould the shoe into its required shape initially. Thus the plasticity gives the shoe its shape and the elasticity ensures that it keeps that shape during its life. It has good tensile strength and stretch properties before it breaks. It allows water vapour to permeate to the atmosphere, but it is resistant to the passage of liquid. The surface can be made into any color required. It is a material easy to work (it can be cut, split, or stitched). Different types of leather can be used in the footwear industry, such as calf, cow, goat, and even snake or crocodile.

The two main types of artificial leather used at present in shoemaking are coated fabrics and synthetic poromerics. The first type of material consists of a thermoplastic coating, such as PVC, or a curable coating, such as polyurethane, over woven, knitted or non-woven fabrics. The second set of material consists of a surface of polyurethane film with either a microporous or micro-cellular layer beneath. These materials have absorption and permeability properties similar to leather. As there is a very wide range of artificial leather materials it is difficult to summarize their general advantages and disadvantages. Several of them lack of some of the more important properties of leather, but some have more abrasion resistance, and the ability to retain better shape. Moreover they are usually cheaper than leather.

The main difference between the leather and its artificial substitutes is that the first one is a natural material, so it is not uniform and its properties vary form one point to the other, while synthetic materials are consistent with defined properties. The thickness and quality of the different parts of skin are determined by the anatomy of the animal. The type of beast skin, the age of the animal, the type of tannage, and the kind of finishing applied to the surface of the skin affects the degree of stretch within the skin. Texture is dictated by the tightness of the fibre structure of a particular skin. The colour of the skin is not uniform, but there are variations of shade. Defects due to natural, mechanical or disease disfiguration which occurred during the animal's life can be present within the skin. Therefore the quality of the skin can vary a lot from point to point.

The action of cutting is usually done by machine using press knives. Two types of presses are usually used; swing beam cutting presses and twin cylinder ones. In the first type of machines the operator places the knife on the material and swings the beam over it and trips the cutting stroke; then the component just cut can be collected and the knife repositioned. In the second type of press the material is fed automatically and a moving cutting head, which holds the cutting knife, cuts the components according to a stated sequence which minimizes the waste of material. While the first type of machine can be used with any type of material, the second one is suitable only for synthetic materials which are uniform and adapted for this type of "blind" cutting.

Therefore people trained to perform the cutting of the leather must be skilled: they need good reasoning ability, as no two skins have blemishes in exactly the same place, spatial perception , as they need to interlock different shapes to achieve the maximum number of cut components from a skin, color realization, as due to the variation of color shades within skins it is necessary to match many cut components, and decision making, as they need to make quick and accurate decisions.

4.2.3 Closing

Closing is the title given to the preparation, fitting, and finishing off the cut components to produce an upper ready for lasting.

The actual set of operations depends of the particular type of shoe produced and varies from shoe manufacturer to shoe manufacturer. What follows is a listing of the most common operations performed on the components of the shoe.

The preparation consists in a series of operations which prepare the upper components for stitching.

Stitchmarks are marked on the upper components to enable the operators to position the sections accurately when stitching. The operation is usually performed by fixing the piece in a jig and imprinting the required marks with a swinging marker board.

Sometimes embossing, which is an ornamental treatment whereby designs are printed into, or raised up on, material by the use of heat and pressure, is performed.

If needed, components are split to a uniform thickness using a band knife splitting machine.

The skiving operation reduces the thickness of certain edges of upper components to:

- improve the appearance of the finished upper;
- to avoid discomfort in wear,
- reduce bulkiness;
- to aid construction.

The reduction of the substance is usually made on the flesh side of the material leaving the grain side complete (sometime this one is skived as well to allow an adhesive to penetrate where two parts are to be stuck). Thinning the flesh side of the component does not reduce its durability or weaken its strength, as the plane of main strength of the leather component is located at 0.1 to 0.3 mm below its upper surface [Wil26]. The three most important types of skiving are (see fig 4.1):

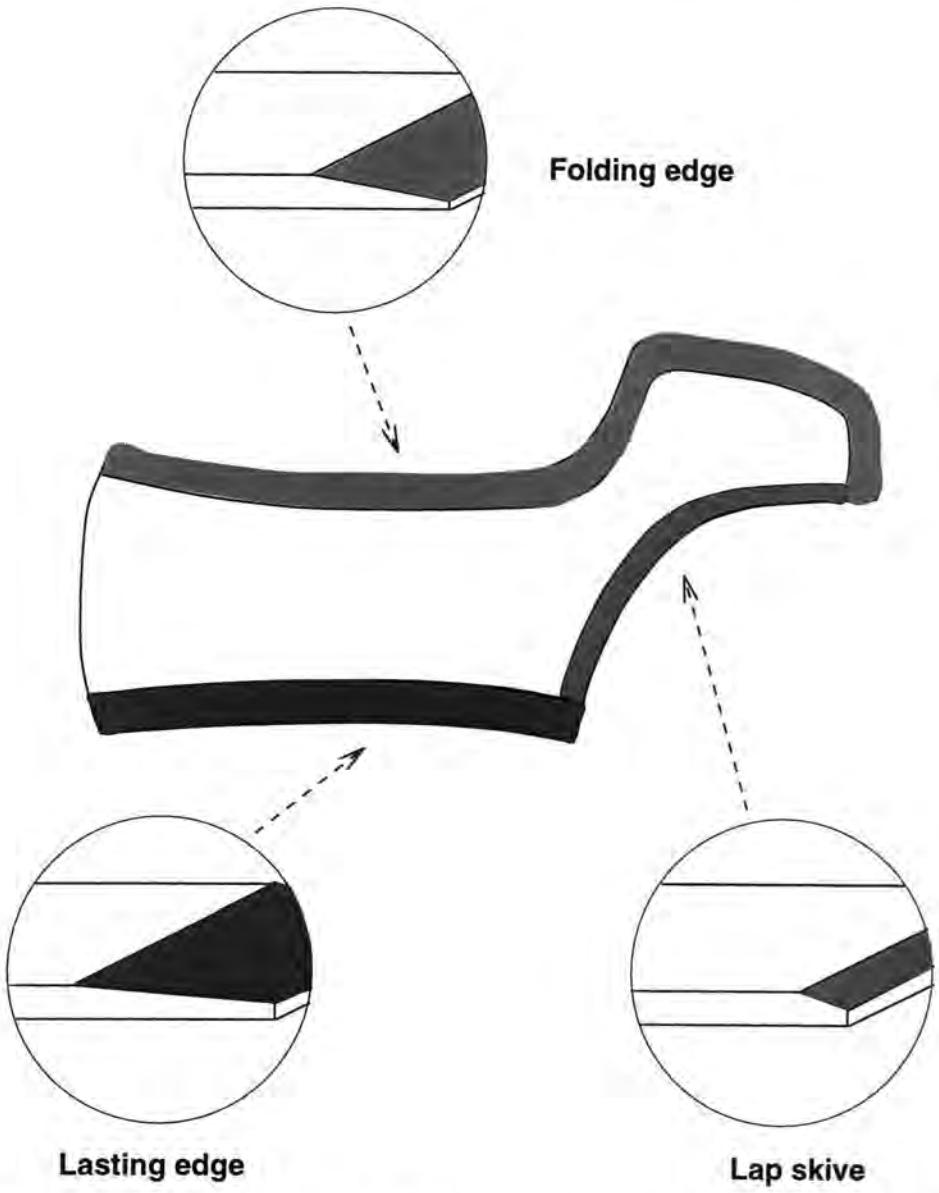


Figure 4.1: Different types of skiving operating on a leather component.

- lapped skive, which is used on the component to be lapped underneath and reduces the bulk which lies against the foot;
- folded skive, where the edge of the material to be folded is skived to twice the width of the required fold so that it lies flat when folded over;
- lasting skive, where the lasting allowance may be skived to reduce bulk beneath the insole.

There are two techniques used to perform the skiving of a component: in the first one the operator manipulates the component by hand through a guide and a sharp

disk knife, so it is skived where intended; it is possible to adjust how deep the skiving should be, how wide, and sometimes the particular skive profile produced. The second technique is called matrix skiving: here the component to be skived is fitted to a rubber matrix which is pre-shaped to conform to the desired end result. The component and the matrix are passed through a splitting machine, which removes substance from the reverse side of the material where the thicker parts of the matrix have applied pressure. This technique is more time consuming in comparison with the previous one, and it is used only when it is necessary to skive small components or complex patterns, where the previous method is unsuitable.

Toplines and edges may be treated in various ways to improve their appearance depending on the style and the price of the shoe. The edge may be left raw as cut or inked to match the color of the surface of the material, or it may be rounded off (burnishing). A binding of leather, fabric. plastic or elastic may be stitched on to an edge. However edge folding is the most common edge treatment for several of the shoe components (see fig 4.2). Once the edge has been skived, it is folded over and cemented down by a manually operated machine, which is similar to a stitching machine; a reinforcing tape may be incorporated to prevent stretching. A more accurate operation can be be performed with a die folder where the component is fitted with a jig, which determines the width of the fold.

The fitting operations mainly involve stitching of one sort or another, and these are performed using several types of different stitching machines each of them developed to perform a particular task.

First, decorative stitching on the leather components is performed to produce nice effects while the components are still flat. Then the backseam is closed (the two pieces of leather which comprise the back of the shoe are joined together) and some backseam tape is added on the stitched area in order to reinforce the back and produce the appropriate shape (sometimes this operation is performed before folding). Subsequently the linens and the other components of the shoe are stitched together in the appropriate order. Finally the shoe is closed and its gains a three-dimensional shape.

During the fitting operations certain parts of an upper may be reinforced to

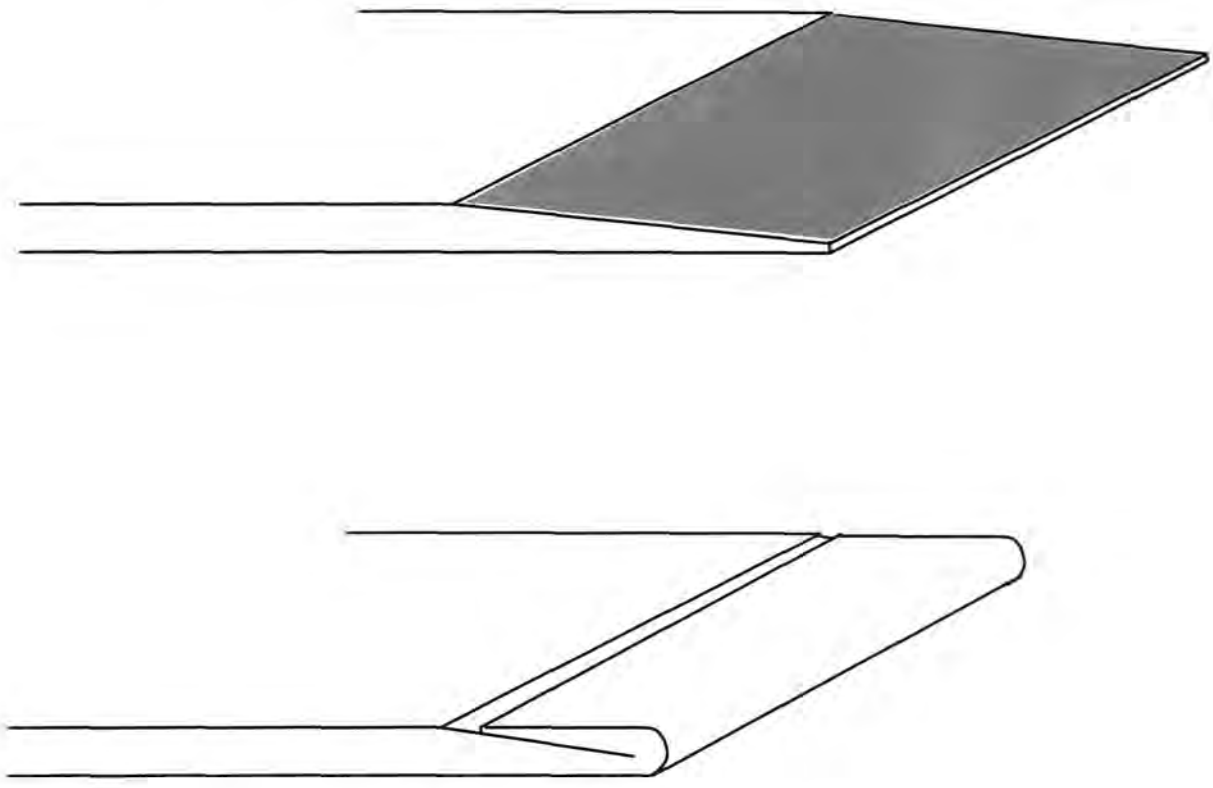


Figure 4.2: Folding operation.

give strength, prevent distortion by stretching, give the material substance and aid waterproofing. It is normal, for example, to use tapes along seams, and puffs to reinforce the toe areas.

The operations involved in the finishing off section vary a lot. Eyelets, for example, are inserted at this point.

Making

Making is the name given to the department in which all the component parts of the upper and bottom are brought together to construct the shoe.
The first operation consists in placing the upper on a last, in order to give shoe the required shape, and inserting the insole.
Then the operation performed varies as there are a number of ways in which the footwear can be made. Each particular method is known as a "method of construction" and in each case it consists of a particular sequence of operations allied to a specific method of sole attachment. A list of the most commonly used ones follows:

- direct stuck construction;
- moulded;
- veldtschoen construction;
- machine welted construction;
- california slip-lasted construction;

In the following sections an overview of these method of construction is given, with a special attention for the first two as they are the most important for large industrial production.

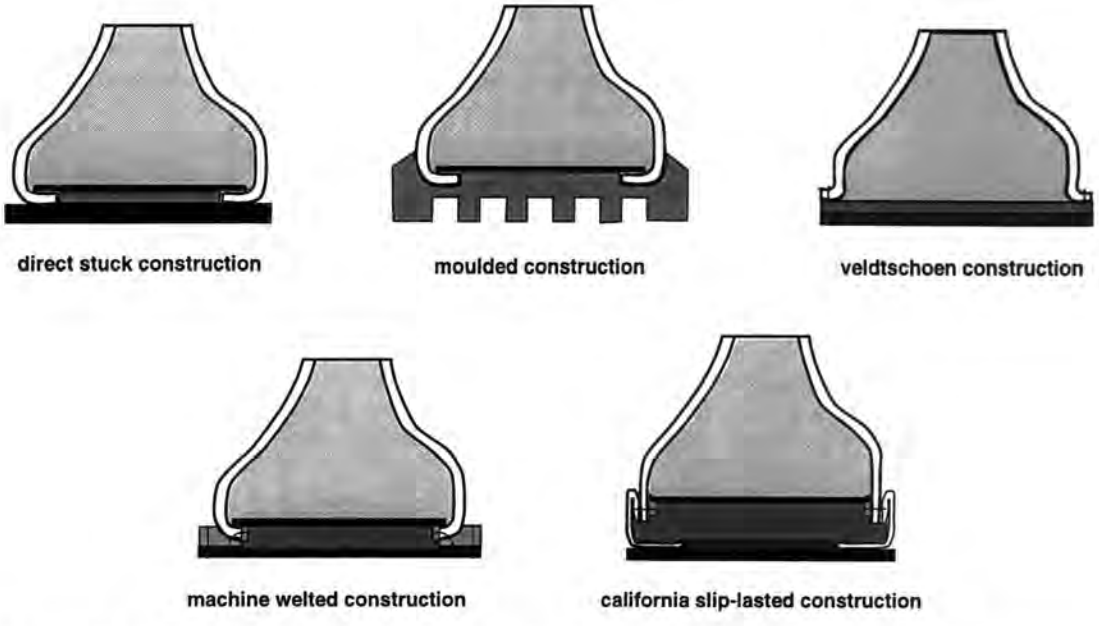


Figure 4.3: Sections of shoes produced using different methods of construction.

Direct Stuck Construction

This method consist of tack-lasting the upper under the insole (with help of some adhesive), and attaching the sole with glue to the bottom of the shoe, and in recent times it has largely replaced the older machine sewn construction.

First a process of back moulding is performed and the upper is pre-heated to soften the thermoplastic coating of the stiffener and the quarters are moulded to

shape. Then after conditioning the foreparts of the last with hot steam, the upper is stretched over the last and secured to the bottom either with tacks or with adhesive. This operation is usually performed by two machines: the first one does the lasting of the forepart(see fig. 4.4), while the second cements the seat and the side of the last. Next the shoes are conditioned again using humidity and high velocity air to eliminate all the stresses and strains, caused by lasting, from upper materials.

Afterward the bottom of the shoe is roughed, an operation that involves the re-

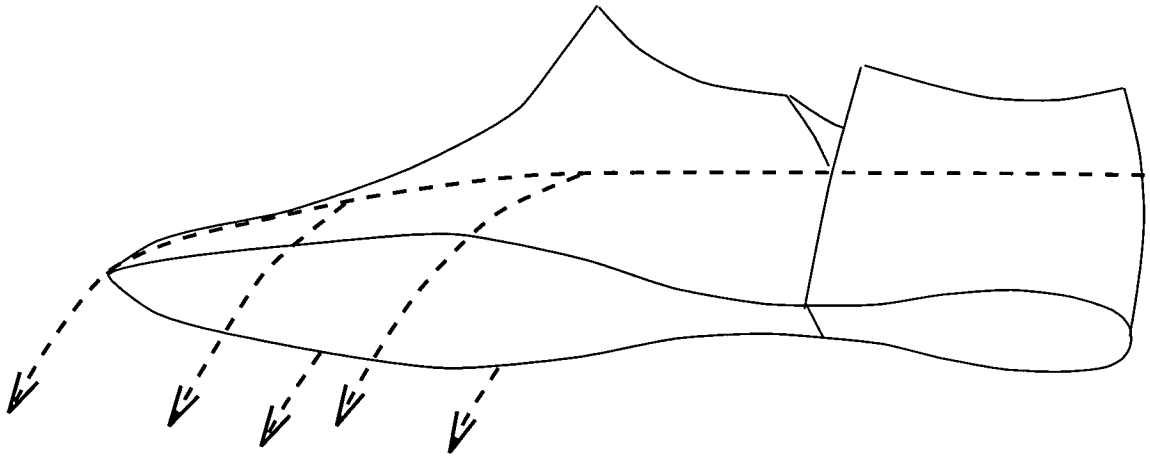


Figure 4.4: Direction of the forces applied to the leather on the last during the lasting operation.

moval of the grain from the complete area of lasting allowance by means of a wire brush or abrasive. Subsequently the adhesive is applied to the prepared area by hand brush, a pressure extrusion, or by a roller type machine. The last is placed in a cement drying and activating unit in order to eliminate all the solvent from the adhesive which reaches the full bond strength intended during this operation. The shoe is then positioned in a sole attaching machine, clamped into position and held under pressure for a pre-set time dwell and pressure. The shoe is cooled, naturally or placing it in refrigerating machines, and the last is slipped either manually or by machine. Finally, the heel is attached using an appropriate machine.

This type of construction is now widely used in footwear industry. Its advantages are:

- economy due to the elimination of skilled operations and the machinery involved with the sewn construction.
- the reduction of operations speeds up the work flow and makes more economical use of the last plant.

Moulded Construction

This construction technique is similar to the previous one, and all the operations up to the roughing of the last are identical. Then the sole of the shoe is produced with an injection moulding of polyurethane or PVC (see fig 4.5). When the moulding process is complete the shoe is removed from the last mould and any flash is trimmed off.

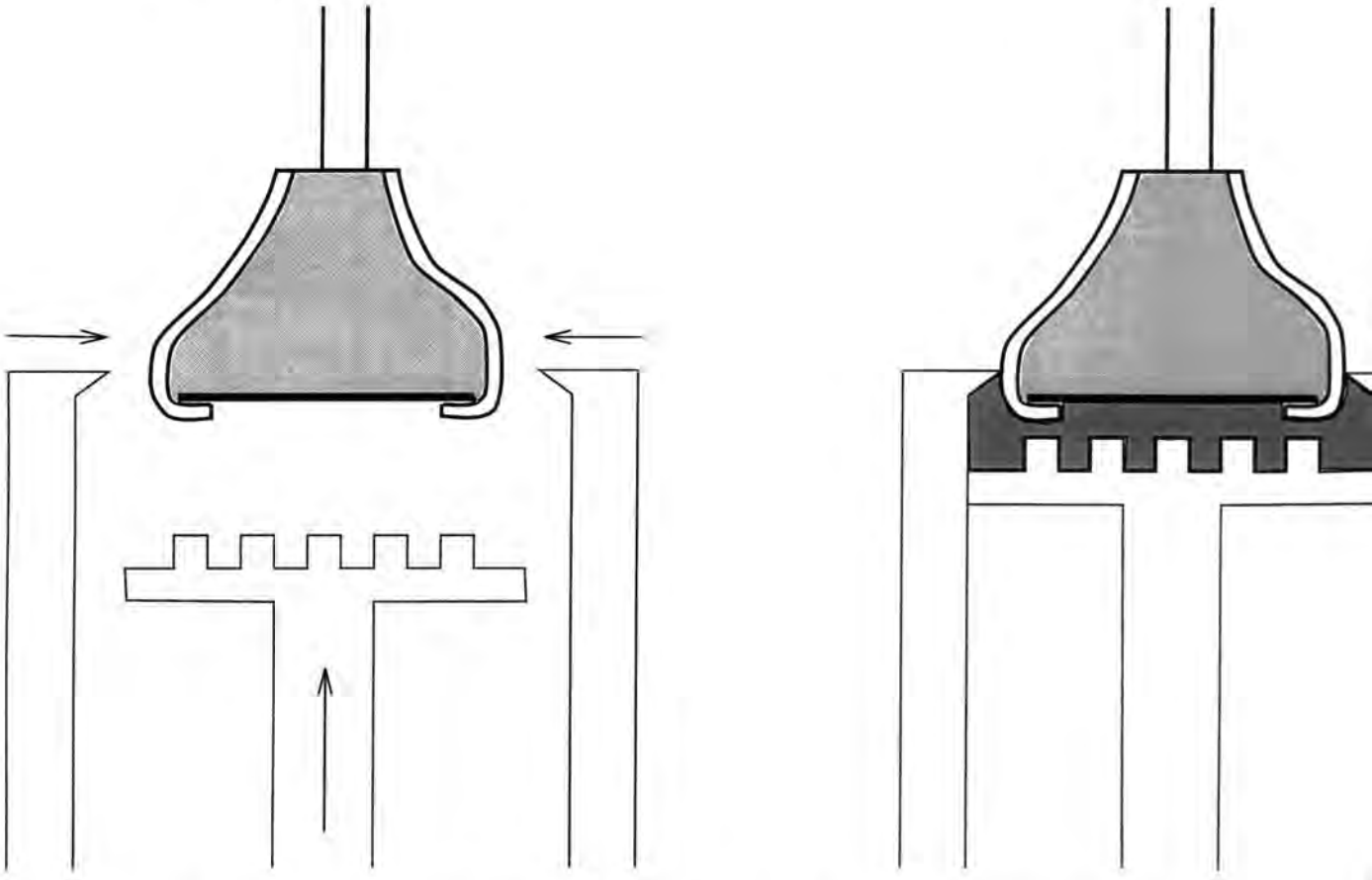


Figure 4.5: Injection moulding of polyurethane or PVC to create the sole of the shoe.

Veldtschoen Construction

The Veldtschoen method is the only one where the upper material is flanged outwards during the lasting process and attached by adhesive and stitching to a layer of material known as the runner or middle. The runner replaces the insole used in other methods. The sole is attached by adhesion. It is strongly recommended for shoes which are to be guaranteed waterproof.

Machine Welted Construction

The welted process was the traditional method of shoemaking: it has lost some of its popularity in recent years, but it is still used for the manufacture of high quality costly footwear. It is expensive because of the high labour content in producing footwear using this method. Further the use of all leather uppers, soles and heels adds to the total cost.

A special insole having a rib or wall is used. The upper is in-lasted to this rib by means of wires and staples or adhesion. A strip of leather, the welt, is sewn to the upper and rib. The welt, having been beaten flat, then has the sole attached to it by adhesion and stitching.

California Slip-lasted Construction

The essential factor to be noted about slip-lasted work is the way in which the upper is constructed in the closing room. This is done in the following manner. A sock, a strip of material (the platform cover) and the upper are stitched together around the feather edge (platform stitching). Then the last is "slipped" into the upper. A coat of rubber solution is applied to the bottom of the sock, and the platform covers are pulled up. Afterwards the operations that follow are identical to the ones used with the direct stuck construction.

4.2.4 Finishing

The finishing operations involve all those activities whose purpose is to give to the shoe a better aspect, such as heel scouring and smoothing, inking of the edges, and cleaning of the uppers. These operations are done mainly manually, with the help of very simple machines.

4.3 Present situation

From the previous description it is possible to understand the reason why the operations involved in the assembly of a shoe are labour intensive and in many cases depend from the skills of the human operator. Some devices which automatically carry out some operations exist, but in a large number of footwear industries a large part of the work is still done manually, and no sophisticated machines are present. However several of the newer machines for the footwear industry are controlled by a computer system which activates and controls all the functions of the device, and they are equipped with a limited number of simple sensors. Only two machines (one produced by BU and the other by ORISOL) use a vision system for operating. These machines are designed to be "stand alone" and they do not require any communication link in order to operate.

Nevertheless some attempts for providing transmission capabilities for improving the performance of some particular machines have been made. The most important effort in this direction has been done by SATRA. This organization in collaboration with all the major shoe machinery suppliers, CAD suppliers, and shoemakers, developed during a period of 5 years, starting form the mid '80s, a set of five different protocols, called InterCIM, especially developed for the requirements of the shoe industry [Pri91] .

Although several different standards for vehicle and aerospace industry had been developed and they were easily available and well documented, none of them could be used for shoemaking as they there not well defined or specified for applica-

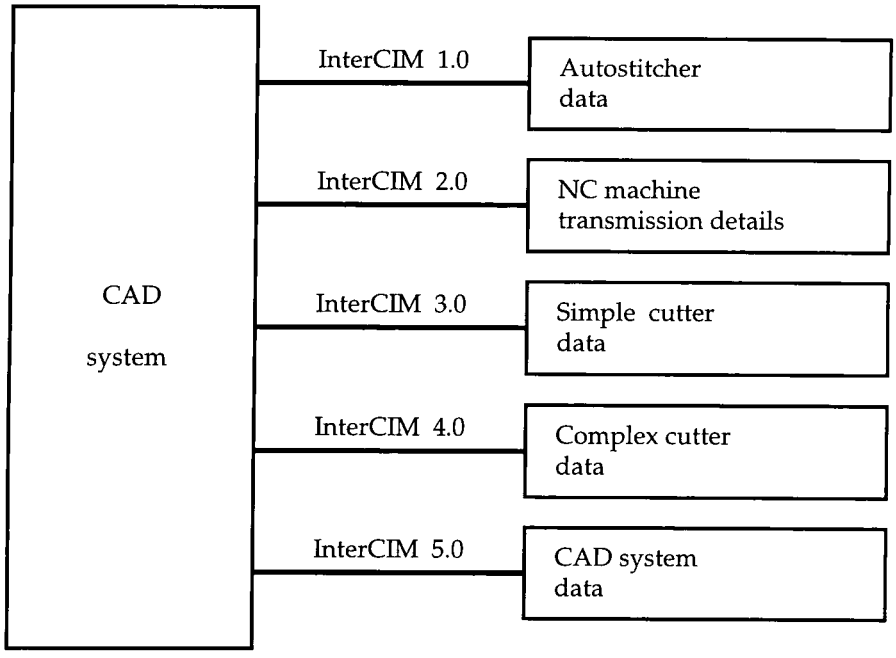


Figure 4.6: Graphical representation of the five InterCIM protocols presented by SATRA

tion in this type of industry. The InterCIM initiative recognized that the available protocols were not suitable for this specific application and decided to adapt the most appropriate of the existing international standards in similar areas, such as RS 274 and IGES, to the specific requirements of the shoe industry. The result has been a series of 5 different protocols which are designed to interface CAD systems, where shoes are designed, with devices (see fig. 4.6).

Their list follows:

- InterCIM 1.0: autostitchers data;
- InterCIM 2.0: NC machines transmission details;
- InterCIM 3.0: simple cutters (plotter-based) data;
- InterCIM 4.0: complex cutters data;
- InterCIM 5.0: CAD systems data.

Details of the five standards are given below.

InterCIM 1.0: "Data format for the transfer of data between CAD systems and numerically controlled stitching machines"

This protocol is used to link CAD systems with autostitchers and it is based on RS 274, a widely used and supported standard employed internationally for driving machine tools in engineering applications. It deals with the form of the data being transferred (what is transferred not how it is transmitted). RS 274 uses a series of standard codes to trigger appropriate machine functions. It has been modified defining some of the spare codes by specify some typical sewing features such as "cut thread" and "back tack". CAD systems should be able to generate the data in the standard form, and the autostitchers capable of interpreting it in the correct way.

In addition, in 1991, an extension of the standard, intended for use with stitchers which use adaptive feedback controls, was developed. Such machines, for example, may sense the edges of the material being sewn, compare the information with that given by the CAD system. They use this feedback to modify the stitch path so that the sewing is accurately aligned with the edges, thus avoiding the need for precise location of the material in pallets.

InterCIM 2.0: "Transmission standard for the exchange of data between CAD systems and numerically controlled machines used in the footwear industry"

This protocol defines how the information between the CAD system and the numerically controlled shoemaking machines is to be transferred. It is based on the RS 232, an international transmission standard, and defines interface connector type (25 pin plug), pin assignment, functional descriptions of electrical signals, word length, transmission speed, and data flow control. The last three items are not included in RS 232 and are added in this standard to further standardize transmission features in footwear industry applications.

InterCIM 3.0: "Guidelines and specification for the use of HPGL plotter control language for driving numerically controlled cutting machines in footwear industry applications"

This standard is appropriate to plotter-based cutters (the ones where basically the plotter pen is replaced by a cutting tool). It is based on HPGL (Hewlett Packard Graphics Language) which was originally developed by Hewlett Packard for their own range of plotters, but it is now used by other manufacturers as well, for both plotters and cutters. Some manufacturers use their own versions specially modified to suit their machines, adding new codes to deal with functions not covered by standard HPGL. The committee found that such extensions were not necessary for simple plotter-based cutters, and a sub-set of the HPGL commands were adequate.

InterCIM 4.0: "Guidelines and specifications for the use of RS 274 data format standard (and its derivative ISO 6983) for driving numerically controlled cutting machines in footwear industry applications"

This standard covers the form of data to drive cutters and, and is recommended by SATRA for any NC cutter machine application other than simple plotter based system (the InterCIM 3.0 is more appropriate for this simple type of machines). Similarly to the standard for the autostitchers it is based on the use of the RS 274 codes to control the machines.

InterCIM 5.0: "IGES sub-set for exchange of data between CAD systems in footwear industry applications"

This standard had been developed for permitting the exchange of data between dissimilar CAD systems. The aim of this system is to provide the possibility for the shoemakers to transmit, for example, 3D data to a sole mould maker, or 2D data of pattern shape to a tooling supplier, even if they have different CAD systems. The protocol had been designed for transferring of geometric data but contains provision for text too. It is based on IGES (Initial Graphics Exchange Specification), a standard used internationally in the vehicle and aerospace industries. A sub-set

approach was chosen and based on that developed for the German car industry, the one that had been judged the most capable of adaptation to the needs of footwear industry. For this particular application the IGES entities have been selected to keep the sub-set as small and straightforward as possible, but a future increase of the codes could be possible.

Although SATRA put a great effort in order to make these protocols accepted by all the manufacturers, several companies still do not implement them in their devices and they have opted for custom solutions.

4.3.1 Other applications

The InterCIM specifications are an important attempt in the direction of introducing communication systems in the shoe manufacturing environment, but they are not the only examples of communication systems specifically developed for this type of industry.

A successful application has been reported in the area of autostitchers [Cla90]. These machines are a class of workcells which bear a strong resemblance to numerically controlled (NC) machines tools. NC machines are usually provided with facilities which allow them to download program files from a host computer. Like NC, machines autostitchers can receive programs via a serial interface, store them in the memory and execute them an indefinite number of times. The advantages of using such a communication system are relevant. Rather than storing programs on EPROM modules or discs, as it is usually done, all the information can be retained in the hard-disc of the host computer and any of the machines connected on the network can access the program files without extensive copying of EPROMs being necessary. The programs can be stored in the memory of the host computer as files written using the InterCIM 1.0 standard language, or a series of binary files which the controller board of the autostitcher can immediately execute. This simple implementation of a transmission system has proved successful when tested.

Another application of communication among machines is developed by ORISOL for one of their leading systems. In this case the CAD data can be downloaded to

stitching machines and gluing machines using a RS 232 link and a proprietary protocol called ORINET. A pilot application have been set up at a Timberland plant in the United States. Similarly Torielli has introduced the possibility of connecting their autostitchers with their CAD system using a RS 232 link and InterCIM protocols. A switch is used to route the connection from one machine to another. In addition the possibility of using modem connection for remote diagnosis is an option on some of the more advanced models.

An example of communication in a different area of shoe making has been developed at BU, where a toe laster had been connected to a seat and side laster. As only one person is necessary to operate these two machines, which perform consecutive operations on the last of the shoe, and which needed repeated commands that had to be typed in, a communication system was established between the two machines so the operator has to enter the information only once. In the first experimental version of the system only information regarding which foot has to be processed and the name of the style of shoe handled is passed, but the protocol of transmission has been designed to be extremely flexible, and therefore other extensions are possible.

An interesting example of an island of automation has been proposed by ACTIS, which has developed a production system for the automation of the first steps of the "making" process. The workcell consists of a set of robots, conveyor belts and storage systems. All the operations are controlled locally and the information for coordinating the activity of all the plant is carried by the supports which hold the lasts of the shoes. They incorporate a small chip which allows the system to identify the component and determine which operations have to be performed on the last.

There are only a limited number of companies which produce devices for the footwear industry which make use of communication facilities. The main reason for this is due to the fact that the devices used on the shop floor are technologically simple. Another reason is that many of the companies offer only a limited number of products in a specific area of shoe production. Therefore they do not have an interest in providing communication capabilities which could be used, amongst

other things, for general monitoring of the whole plant.

4.4 Future directions

The previous section presented an overview of some attempts to introduce communication capabilities among machines used in the footwear industry. It is likely future more and more examples of communicating workcells will find a place on the footwear shopfloor.

At present the majority of shoe industries are not ready for the introduction of a communication system which could link several machines as there are not many machines that could take an advantage from such facility. However information gathered from BU enables an assessment to be made of the capabilities of the most advanced machines produced for the shoe industry and of some devices which will be developed in future. They are controlled by microprocessors, and sensors are used to allow them to perform their activities, which are defined by appropriate programs and therefore can be changed as needed. Such intelligent devices would certainly benefit from a network connection. As designing shoes using CAD systems is getting more and more popular, the CAD data could easily be used to prepare the information necessary to produce the programs needed to control the different machines. A network system would allow the transfer of the information straight from the CAD system to the specific machine without copying the data on a physical device such as a disk or an EPROM.

Prototypes of machines which can perform automatically the stitchmarking [Tou89] and the skiving [Top93] of the components have been built at Durham University. A machine which will be able to produce the cutting knives using CAD data is under study at BU. All these application depend extensively on CAD data in order to perform correctly their tasks in order to avoid, or at least considerably reduce, the "teach" sessions. These are sessions where the machines are programmed to perform a specific set of operations. To obtain an easy and fast way for the transmission of the necessary information a network system would be extremely desirable.

From the previous discussion it is clear that the type of information transferred is mainly related to CAD data which is used to produce the control programs for the different machines. However other types of data can be transferred: for example remote controlling and supervision of the plant would be possible. The machines could send each other information relevant for performing their tasks, such as the style of shoes currently in production, so they can set them up properly without the help of a human operator. Data from sensors which cannot be processed locally could be sent to other stations, or this data could be sent to other machines for which it could be useful. Messages for indicating an abnormal operation and the instructions for recovering from it could be issued to the interested devices. Monitoring of the resource use and device utilization is another opportunity offered by a networking system. Finally a communication system could satisfy the growing interest for tools which enable remote diagnosis of industrial machines.

The communication system in this type of application is not a strict real-time one as the information transferred in usually not immediately necessary to the machine in order to perform the task, and it is possible to think that small delays are acceptable. This particular feature allows the design of a simpler system which do not have to cope with the problems of the communication systems used in strict real-time applications where delays could be not acceptable.

4.5 Manufacturing cells in the footwear industry

The aim of a communication system in a manufacturing environment is to provide communication means between all the connected components. The complexity of the system may vary, but in complex applications it is possible to identify always three main building components. The first is the database which consists of program files encoding the operations that the machines have to perform on the processed items. The second is the production management system which supervises the whole production, keeping track of the components in production and controlling the contents and the activity of all the devices and in some cases deciding the production schedule. The third include the devices which represent all

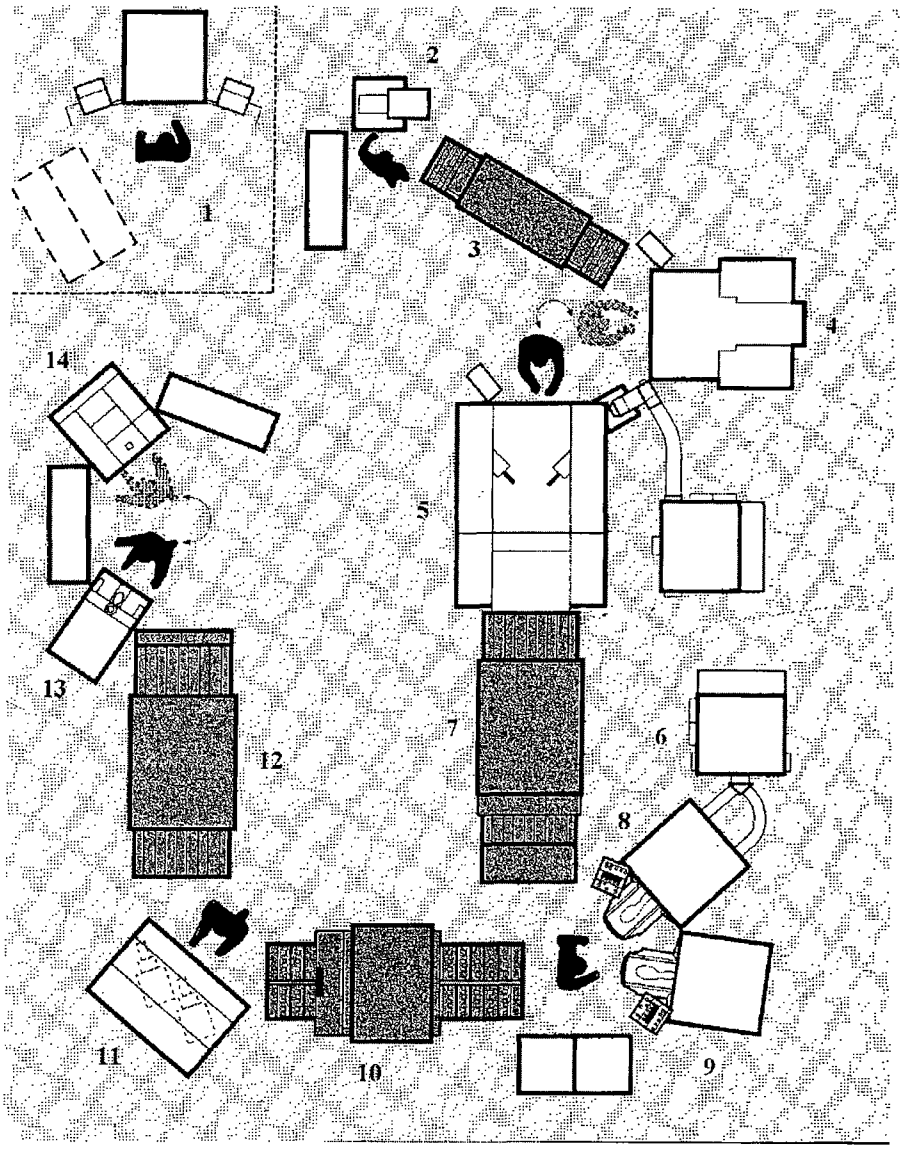


Figure 4.7: The BU "Rink System" cell layout for the the lasting and bottoming of the shoes. 1: backpart moulding, 2: insole attach 3: upper conditioning, 4: forepart lasting, 5: seat and side lasting 6: dust extraction 7: heat setting, 8: auto roughing, 9: auto cementing, 10: cement drying, 11: sole attaching, 12: shoe cooling, 13: last slipping, 14: heel attaching

the other units connected to the communication system.

In the specific case of the the footwear industry a list of possible devices which could find place on the plant floor follows:

- computerized numerical controlled machines: they represent majority of the machines used in the production line (autostitchers);

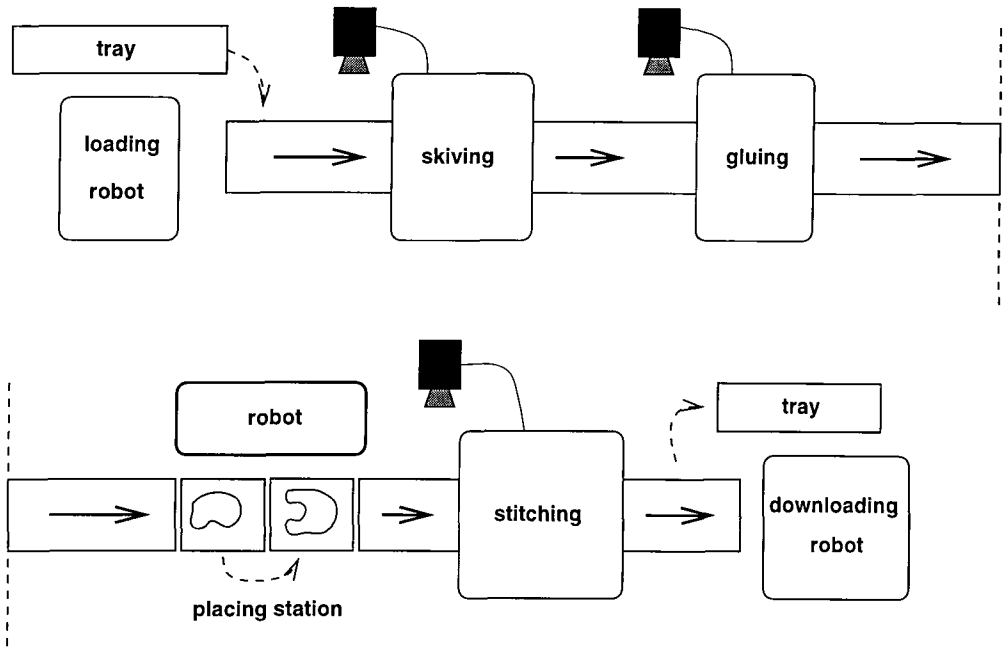


Figure 4.8: Schema of the production line for the assembly of the uppers of shoes

- dedicated devices; they include broadly all the set of devices which have been developed essentially for the footwear industry. The cited skiving and stitching machine are examples of such devices, but many other ones are present in other areas of the shoe assembly;

- industrial robots: there are only a few example of usage of robots in this area, but they will be useful for performing some operations in the last part of the assembly of the shoe, and in the handling of the materials;

- material transport systems; conveyors belts are actively used for the transport of the material along the production line; an eventual use of AGV should be considered;

- automated storage systems: they will be essential for the automation of the production plant as it is likely that the components of the shoes will be processed not in logical order but in sets of similar pieces (semi-random);

- programmable logic controllers: as they are in general use as control mechanisms they will be useful in several applications for controlling other simple devices;

- bar code readers: they are used mainly for identification purposes;
- vision systems: used for the identification of the pieces and for the inspection of the correct assembly of the components;
- sensors: used for the identification and inspection all along the production line;

An integrated system must be able to supervise and control workcells which can be composed of any of the cited devices.

4.6 Two examples of networked workcells for the footwear industry

An existing example of devices, which can be set up as a workcell, and which could take advantage from the introduction of a communication system are the ones in the BU "Rink System". The devices in this system perform the operations involved in the lasting and bottoming of the shoes. They can be used as stand alone machines, but they can be placed in a configuration which allows a continuous production flow. Such a disposition is presented figure 4.7. In this configuration information on the style of shoes processed could flow from one end to the other of the rink reducing set up time. The two machines which have been experimentally connected together, as previously described, are part of this system, and the advantages provided by this link could be easily shared by other machines as well.

Another area where there is the potential for creating an island of automation in the production process can be located in the first stages of the assembly of the shoe. Several dedicated devices have already been developed, and during this research it has been possible to design a support infrastructure which allows them to become integrated in a manufacturing cell. The layout of the proposed workcell is presented in figure 4.8. The aim of the system is to automate the first stages of the assembly of a shoe by taking as input a cut shoe component and producing

as output embroided and/or joined components. A brief description of the devices follows:

- skiving machine: it operates using the principles of the skiving machines developed in the past at Durham University;
- gluing machine: it is basically a NC machine (autoscan type) which has a glue dispenser instead of a needle. A preliminary study for the implementation of a gluing system using electrostatic glue deposition is presented in chapter 5;
- stitching machine: an appropriate version of the Autoscan developed by BU;
- placing robot: for placing together components which need to be stitched together;
- robots: they are necessary to carry out the loading and the unloading of the components which have to be processed.

The components to be processed are placed in batches in an input tray. Research done at Durham University on devices for the footwear industry started from the assumption that the components to be processed by the machines were to be processed in random order. However from visits in shoe factories it has been observed that this is not exactly the case since the component are usually batched in sets of twelve. Therefore it is possible to assume that some sort of batching is present at the level of the input storage system. The production system, starting from its knowledge of the type and the number of components present in the input tray, decides the production schedule. The loading robot places the single component on the conveyer belt. Then the skiving of the component is performed. Then if it is necessary to stitch together two (or more) components, glue is applied to the lower ones and they are placed one on top of the other by a placing robot. This operation is performed in order to avoid relative movement of the components. Components which do not require the joining operation skip the previous operation when passing through the system. Finally the stitching operation is performed before the component is downloaded to the output bay.

From the figure it is possible to see that several of the systems use vision systems. The Autoscan stitcher produced by BU actively uses the vision system in order to identify the components. A communication link is able to inform the devices which component to expect next, but the need for a vision system remains since it is necessary to determine the exact position and orientation of the components. At the present unfortunately there is no available data of the drifting of shoe components on conveyor belts, and it is possible to assume that if their movement is negligible some of the vision system could be eliminated, or simpler ones implemented.

These two examples of feasible workcells allow us to define the transmission needs in the footwear industry and show the possibility of creating islands of automation in the shoe production process. A complete integrated communication and control system able to support these workcells will be developed in the next chapters.

4.7 Conclusions

Despite a lot of work related with computer networks, the area of industrial communications has not been yet fully developed. In particular in shoe manufacturing almost no implementation of communication systems exist, despite the obvious advantages that such infrastructures would have in the manufacturing environment.

The existing communication protocols which have been specifically developed for the needs of the footwear industry have been described. However they mainly establish a format for data representation for the computer controlled devices used in this manufacturing area, and they have not achieved a wide acceptance.

The current directions in which the footwear industry is moving have been reviewed in the chapter. It has been possible to see that the level of complexity and intelligence of the devices is gradually increasing and sophisticated features are being included in the latest models. In this environment where elaborate devices are starting to appear, the benefits of a communication link are evident. Data transfer from CAD systems to the devices, coordination among machines,

better understanding of the production flow and powerful tools as remote diagnosis and software updates could become possible. The last section of the chapter has described two examples of manufacturing cells which could take advantage of a communication facility. The first example consisted of an existing cell used in the area of the lasting and the bottoming of the shoes. An experimental limited communication link had already been tested between two of the devices included in the cell, and this experiment highlighted the type of advantages that the introduction of communication facilities could offer to the whole system. The second cell is based on the integration of the devices which have been developed in the recent past at Durham University, Hull University and by BU, and represents an example where communication facilities are desirable in order to implement efficiently the automation of an assembly process. In particular a preliminary study for the implementation of a gluing system, the key element for allowing the joining of components in the proposed cell and the only one device in the cell which has not yet been developed will be presented in chapter 5.

It is possible to conclude that it is likely that future computer controlled machines will depend on the use of a communication system linking them together. Even if the idea of creating a completely automatic shoe plant is not realistic, the use of the communication links will produce benefits on all the production operations, even where the human skills are necessary, and will permit better supervision and control on the plant.

Chapter 5

Electrostatic glue application for footware applications

5.1 Introduction

In the shoe making process it is required that flat pieces of leather are attached together, in a specific orientation with respect to each other, before they are stitched in an automated sewing machine. This process is currently done automatically using a template board and punched holes in the leather components. The first component to be glued is placed onto the template board aligning the pins through the "guidance holes" which are pre-punched into the leather. The second component is carefully "dabbed" onto a tape dispensing glue, which deposits a "tacky" blob onto the surface of the leather. This component is then aligned onto the first one, using other aligning pins and guidance holes, on the template board. Then the two components are pressed together to ensure a bond. A separate template board is required for each design of shoe, and also for various sizes. The manual process is very time consuming and an automated method would be ideal since it would allow the automatic assembly of a consistent part of the shoe uppers without the use of manual labour.

This chapter examines the various possible methods of depositing glue onto the surface of leather prior to stitching, with the idea of incorporating the most effec-

tive method into a fully automated process. This should ensure the deposition of a sufficient amount of glue on the overlapping areas to ensure the bond, and the position of glue should not interfere with the stitching path in order to avoid problems with the sewing needle. Since the system will be included in an automated process the system must be able to cope with components randomly ordered and with any orientation and position.

Due to the large number of possible designs and sizes of the shoe components a computer controlled mechanism for the deposition of the glue is required. Research on a related problem has been carried out by Nigel Tout [Tou89] when he investigated the possibility of automating the stitchmarking operation. This consists in marking the upper components of the shoes with a pencil, or another marking device, in order to allow the correct positioning of the components before manually stitching. In our case there is the additional constraint represented by the glue involved in the process.

The most direct approach in order to achieve the deposition of the glue on the leather components is to carry on an analysis on the currently available systems for printing on paper and to analyze the possibility of modifying them in order to make them capable of transferring glue onto leather. The computer controlled printing mechanisms can be grouped into two sets; small-area markers and full-width markers. The first set includes all the systems which can apply only a single point at any time, thus lines are drawn by sequentially moving the writing head across the paper. The second set includes all the devices which can print more points simultaneously and the printing action is achieved using the relative motion between the paper and the printing line in one movement.

Both groups of printing mechanisms are analyzed in the following sections and the possibility of adapting them to the automatic deposition of glue is discussed.

5.2 Small-area markers

Small-area markers use a device which can generally mark only one point at time. Thus, in order to draw lines, the printing device has to be scanned raster fashion

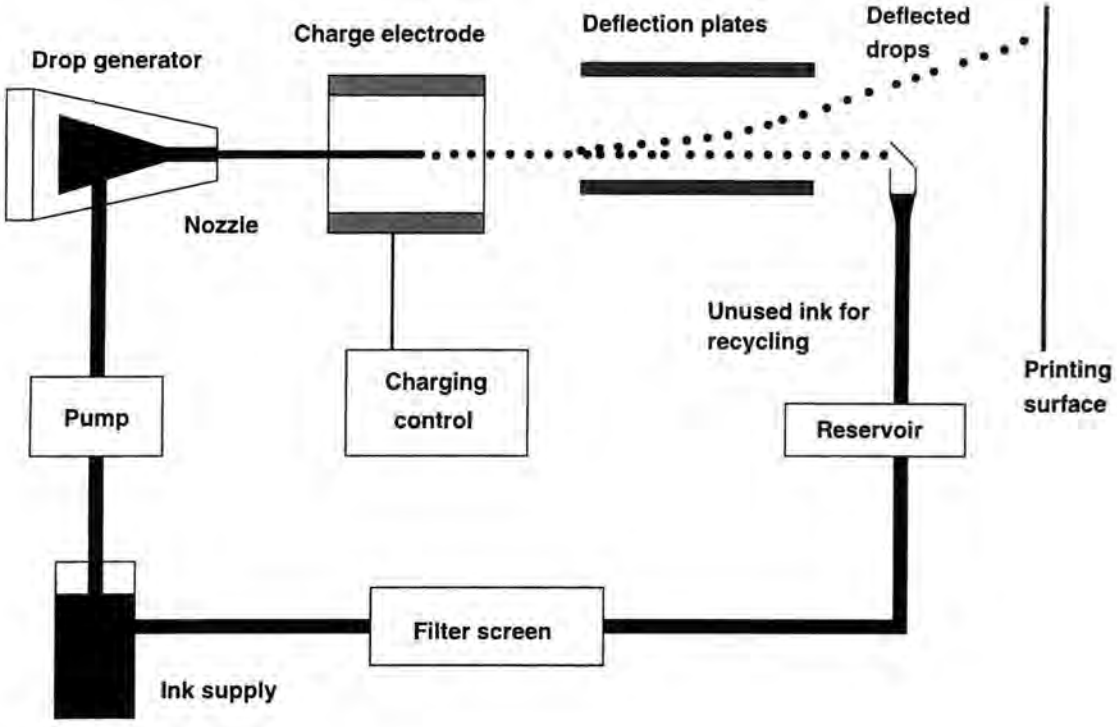


Figure 5.1: Inkjet printing fires droplets of ink which are inductively charged and then deflected with an electric field.

over the component and activated when it is over a spot which has to be marked, or it has to be used as a X-Y plotter and trace along the desired patterns. In both cases the motion can be achieved moving the printing device or the paper being marked. The devices grouped in this category include pens, dot matrix impact print heads and ink jet devices.

It is possible to analyze the possibility of adapting these printing techniques for the deposition of glue on leather surfaces.

The easiest approach is to adapt the pen printing technology used in X-Y plotters. It is possible to think of designing a device similar to a pen which applies glue in the appropriate positions. The use of liquid glue or hot melt glue would be possible with suitable devices. The advantage of this approach is that it is quite cheap, and it uses a simple control mechanism. On the other hand these modified pens pick up dirt which increase the accumulation of glue near the nozzle which may eventually obstruct the glue source. If the glue is too liquid it may drop on an undesired position. The major disadvantages are that since they work by contact, a method must be used in order to hold the component in place while the device

applies the glue over the leather.

The deposition of glue using its intrinsic electrostatic properties after having placed upon the leather an electrostatic charge has been proposed in an undergraduate project at Sheffield University [Cul96]. The system would resemble an X-Y plotter where the pen has been replaced by a electrostatic charging device. After the application of the charge on the leather, the powdered glue would be deposited and attracted to the exact position of the charge. The proposed system has the advantage that it eliminates the problems related with the obstruction of the glue source. It is not possible to estimate the speed of the system without further investigation, but it is likely to be quite slow. Special safety precautions are required due to the high voltages needed to allow the deposition of the charge on the leather. Lastly the glue will stay in place as long as the charge on the leather remains and there are chances of contaminating unwanted areas of the leather with uncharged glue.

Dot matrix prints heads consist of linear arrays of 7 to 30 fine wires which can be individually controlled and fire upon request against an inked ribbon suspended close to the surface to be printed. It would be easy to adapt the process to the deposition of glue if the inked ribbon were replaced by tacky tape (produced by 3M). The advantages of the system is that it would be cheap to implement, easy to use, it would require no heating system and it has already been proven that it works since it is used in the manual process. The disadvantages include the lack of accuracy in comparison with liquid or powder technologies, the gluing head could be only used in a raster pattern over the component which would have to be held firmly during the printing operation.

Inkjet represents a non-contact printing technology where a small nozzle issues a stream of ink droplets which is directed against the surface to be printed. The position of impact of the droplets is usually controlled by electrostatic fields which deflect the ink stream. These are usually continuously cycling devices, where droplets are produced continuously, and the ones which are not needed are collected for recycling before reaching the surface (see figure 5.1). The problems related with this technology are that glue tends to block nozzles and that it is completely un-

suitable for recycling since if it solidifies it would block all the ducts. In addition there is the problem related with the contamination of the nozzle with the dirt and the debris of the leather. This technology cannot be used in situations where the device is subject to accelerations, so it must be kept stationary while the surface which is printed moves perpendicularly to the direction of the deflection. However some devices exist, which produce ink droplets only when required (called "drop on demand") and are usually based on non-deflection mechanisms. The ink stream is modulated by electrodes and must use relative motion to produce printed images. However if glue is used, the interruption of the flow may cause the blocking of the nozzle, making the system unusable. Several of the problems of the use of inkjet technology for the deposition of glue could be solved if a liquid glue which is not sticky before activation could be employed.

The possibility of adapting small-area markers technology for the deposition of glue offers different approaches which could lead to a successful implementation; however, in all the cases the main problem is related to the fact that either the printing device or the shoe components or both have to be physically moved around. This implies poor processing times whenever a raster or X-Y plotting technology is used, and loss of precision due to the movement of the component while the glue is applied.

5.3 Full-width marking

The main advantage of the full-width markers in comparison with the small-area markers consists in the fact that since the former can mark simultaneously in many places along a line, the component can be printed in one pass if it is moved perpendicularly to the printing line. This results in an reduction of the processing time and in an increase of precision during the operation.

Devices which are classified as full-width markers include inkjet devices and thermal printers. The first group uses arrays of inkjet nozzles which cover a small part of the whole width. The problems of applying this technology for the glue

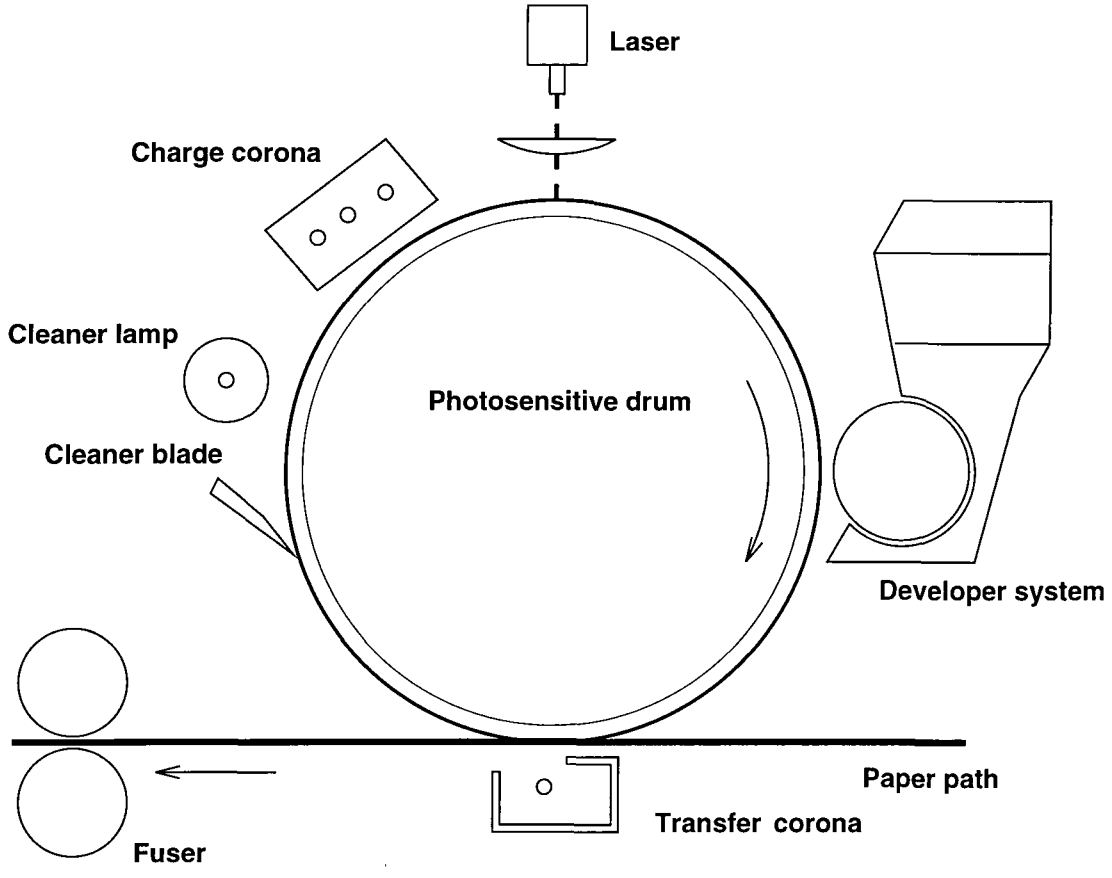


Figure 5.2: Schematic cross section of a electrographic printing engine

deposition are the same described in the previous section for the normal inkjet devices. Thermal printing is based on a bar of resistance elements in contact with a waxed carrier which touches the surface to the marked. The image is produced by melting dots of wax which are transferred from the carrier to the surface being printed. The main problem in applying this technology to the deposition of glue is that when the glue is melted it becomes sticky and therefore the process would require extra care.

The largest group of devices which are classified as full-width markers are based on the electrographic process which offers several advantages for the deposition of glue in comparison with the techniques described so far. An overview is presented in the next section.

5.3.1 Electrographic print mechanisms

Electrophotography is the technology which is the base of virtually all photocopies and laser printers. It is a complex process involving, in most of the cases, six distinct steps which are shown schematically in figure 5.2 [She92]:

- **charge**: a corona discharge caused by air breakdown uniformly charges the surface of the photoreceptor which, in absence of light, is an insulator;
- **expose**: light, reflected from the image (in a copier) or produced by a laser (in a printer), discharges the normally insulating photoreceptor producing a latent image which mirrors the information to be transformed into a real image;
- **develop**: electrostatically charged and pigmented polymer particles, called toner (about 10 micrometers in diameter), are brought into the vicinity of the latent image. Thanks to the electric field created by the charges on the photoreceptor, the toner adheres to the latent image, producing the real one;
- **transfer**: the developed toner on the photoreceptor is transferred to paper by corona charging to the back of the paper;
- **fuse**: the image is permanently fixed to the paper by melting the toner into the paper surface;
- **clean** the photoconductor is discharged and cleaned of any excess toner using corona lamps, brushes and scraper blades.

Each of the steps can be implemented using different technical solutions and almost every vendor has developed some proprietary techniques in order to overcome the problems associated with each of the steps of the process. Several types of photoreceptors based on various chemical components with different spectral responses have been introduced in order to increase the life and decrease the cost of the photoconductor drum. The corona discharge method is a major source of reliability problems since it produces corrosive ions and other reactive components which can

damage the photoreceptor as well as the corona wire itself. In addition it is necessary to clean it periodically from toner and paper dust without breaking the thin corona wires. Canon introduced a charge roller [NHAK89] which decreases the problem of the ozone production, but more work has to be done in order to find an alternative to the present systems, which will have longer life, charging uniformity, and lower cost. The light source for impressing the photosensitive drum is generally a GaAlAs semiconductor laser, but some printers which use LED arrays have been introduced, which allows a better reproduction of the image since no horizontal scan skew errors are produced. There are three types of development systems used; two component, monocomponent and liquid. The two component systems are usually used for high speed machines. The developer mix is composed by two components, the toner, with a diameter of about 10 μm and the carrier which is made by particles of about 200 μm. Several variants of two-component systems are employed in commercial printers. These devices are quite complex, and only a part of the toner is transferred to the paper. Conceptually simpler are monocomponent systems where the carrier particles are eliminated. There are three independent choices to be made in the system:

- the toner could be conducting or insulating;
- the toner could be magnetic or nonmagnetic;
- the toner could contact or jump across the gap to the photoreceptor.

Due to the simplicity of the monocomponent systems, they have fewer parts, smaller hardware, and lower manufacturing costs in comparison with two component ones. Some of the problems are related with monolayer development (hence gray copies) and humidity sensitive transfer. In both systems the toner must be charged so that the electric field of the latent image can attract it on the appropriate areas. Additives called "charge control agents" [Gru87] increasingly are being added to the toner to control its charge as wrong-sign toner can produce background, uncharged toner produces dust, and the average charge-to-mass ratio determines character and solid area optical densities. A surprisingly large variety of charging methods,

especially for monocomponent systems, have been identified and incorporated into development systems. Examples are induction charging, contact electrification, or corona charging (see figure 5.3). The liquid development process has the advantage that the hardware necessary is simple since no mechanical transport device is used and no fusing is needed, however it has the problem of the retention in the paper of the solvent which is later released into the environment.

The simplest implementation of electrostatic printing for the deposition of glue can be seen as a generalization of the system proposed at Sheffield University. It consists of an array of charging electrodes very close to the leather surface, with a conducting plane behind. When a potential difference is applied between the electrodes and the ground plane an electric charge is deposited on the leather surface. The latent pattern can be made visible by a developing system where the charged glue particles are transported into the vicinity of the charged areas to which they become attracted. The main problem related with this application is that the dielectric medium used to receive the charge from the electrodes must have a time constant sufficiently long so that the charge does not dissipate before the toner is transferred. Dry leather has semi-discharge time of the order of minutes, but it become of the order of few seconds as soon as the humidity level rises above 50% [MMCC95].

A laser printer could be used to print a computer generated glue pattern if powdered glue could be used instead of the toner. In order to make this possible it is necessary to use a glue whose particle dimension is comparable with the dimensions of the toner. The glue must be not "sticky" when it is in its powdered form, and it must be possible to activate it later in the process. The fusing mechanism of the laser printer needs to be disabled otherwise fusion of the glue with the resulting contamination of the engine of the printer would occur. Since the glue is naturally a nonmagnetic insulating material, adapting a laser printer which uses a monocomponent development system with toner with such characteristics represents a direct way for achieving glue deposition. The major concern about using laser printers for glue deposition in a shoe factory environment is the uncertainty about the working life of the photoconductive coatings which are generally

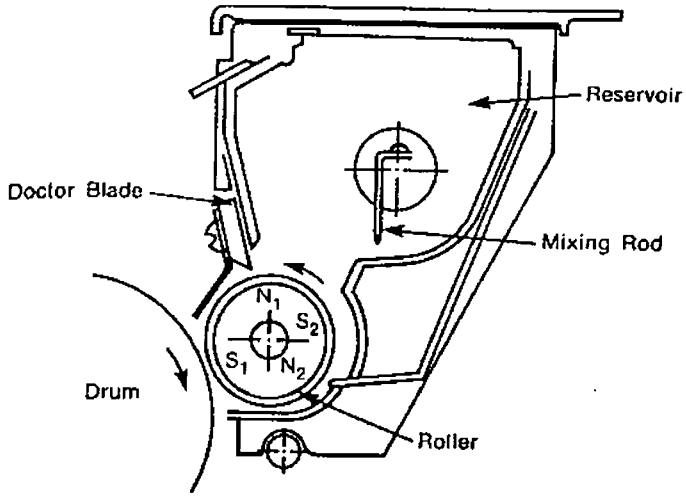


Figure 5.3: In the Canon monocomponent development system the magnets are stationary and the toner, containing magnetically soft material, is carried by the roller past a magnetic doctor blade into the development zone

based on selenium or organic resins. They are fairly weak physically, so may wear quickly when used with shoe materials and could be damaged by staplers or tacks which might pass through. They are also prone to contamination by compounds in the shoe components. In addition it is necessary to lay down on the component a suitable layer of glue in order to create an effective bond when the components are put together.

For glue deposition, electrographic processes have the advantages that they are very compatible with digital electronic systems, they can print patterns onto surfaces which are moving past at steady speed, thus making easier to include them in an automatic assembly system, and they do not have the fine nozzles of inkjet printers to become blocked. Therefore they are good candidates for the implementation of an automatic glue deposition system even if they present some weakness due to the fragility of some subsystems.

5.3.2 Ionography print technology

A type of electrographic printer which has the advantage of robustness and high speed is the ionographic printer, which has been developed by Dennison Manufacturing Company in conjunction with Delphax Systems. This technology, repre-

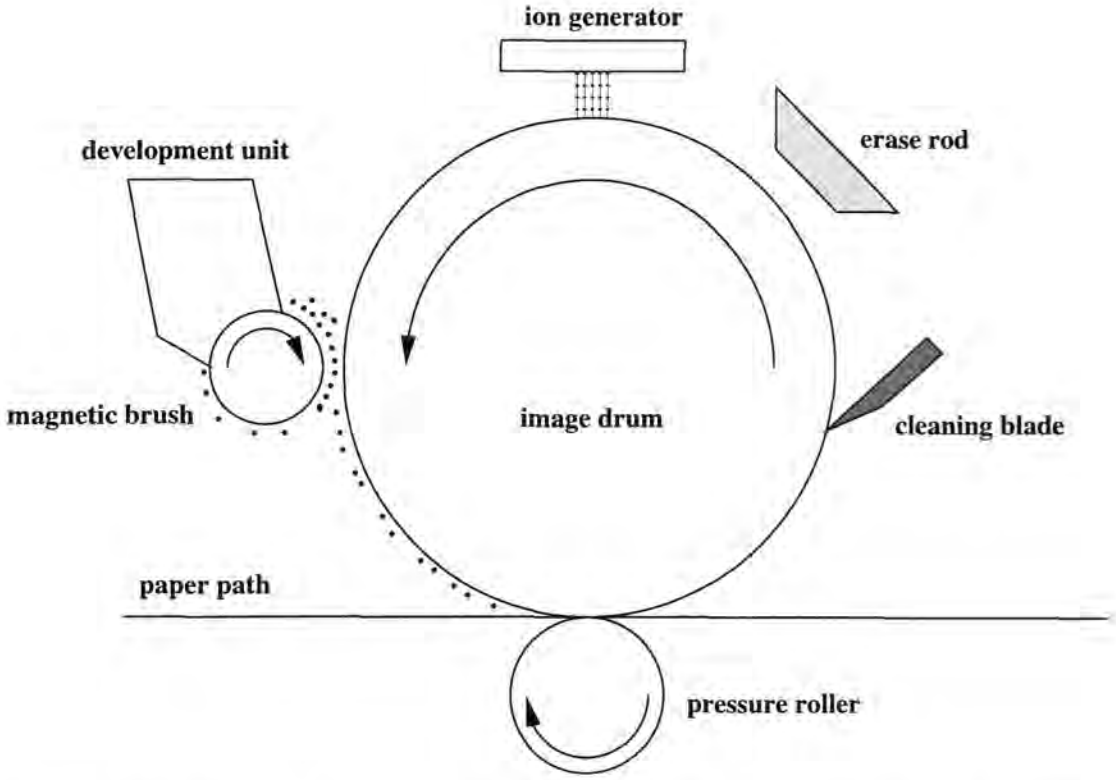


Figure 5.4: Schematic cross section of the Delphax Ionographic print engine

sented schematically in figure 5.4, employs an offset method of electrography using a a dielectric drum made by aluminum covered with a very hard anodized coating, about 40 μm thick [RB86]. The latent electrostatic image is created directly on the surface of the anodized layer by a closely spaced ion generator which produces jets of negatively charged ions from a series of fine holes along its length, which allows an accurate generation of the image. The development system uses monocomponent conductive magnetic toner. The conductive toner transfer problems with high relative humidity have been solved by combining the transfer and fusing steps into one. A pressure roller underneath the aluminum drum, between which the paper being printed passes, squeezes the toner powder into the paper fibres, which have the effect of transferring an fusing the toner simultaneously. The cleaning of the hard dielectric is easier to implement in comparison with a soft photoreceptor. Delphax machines physically scrape off the remaining toner with a steel doctor blade, and then erase the remaining electrostatic image with a corona device.

By eliminating the photoreceptor, and therefore the charging and exposure steps, and using a different transfer method, this represents a simpler process.

Furthermore the harder receptor surface leads to improved reliability.

The main problem for adapting the process for the deposition of the glue on the leather components is related with the high pressure transfer technique which may damage the leather components and would certainly fuse the glue with obvious problems. The only solution would be the removal of the pressure roller and the introduction of a normal coronatransfer [Sch75] which would allow transfer without fusion of the glue.

Another relevant problem connected with this technology is the fact that Delphax, the main producer of this type of printers, has already been contacted in order to develop their technology for the stitchmarking of the shoe upper described in Tout's thesis [Tou89], but they did not show great interest in licensing their technology to BUSM. Therefore it is likely that this problem will occur again if a new proposal of collaboration is proposed. In addition their machines are designed to work at extremely high speeds (up to hundreds of pages per minute) which are unnecessary in the proposed application. This in conjunction with the necessity of replacing the pressure roller system, and the possibility of using this technology for stitchmarking, would make the development of a ionographic system suitable for these applications a rewarding research area for BUSM.

5.4 Experimental work

From the previous overview it is possible to observe that full-width printers are mechanically simple and they are able to print continuously with the components travelling at constant speed. These factors give the advantage of reduced maintenance, and simpler control in comparison with single point or small-area markers. Therefore they represent good candidates for a re-engineering process.

The analysis of several types of electrographic printers showed that the ionographic printers use the simplest and most robust technology. However their engine needs to be accurately modified in order to be adapted for the deposition of glue on the shoe components. This in conjunction with possible problems related with technology transfer issues from the main producer of ionographic printers suggested

to employ laser printers for building a prototype glue deposition system. A particular attention was placed to printers which use monocomponent development systems.

The first step for the development of a prototype for glue deposition using a laser printer consisted in analyzing the feasibility of using a normal laser printer for printing glue on leather surfaces. The operation of transferring toner to leather using a normal laser printer has already been proven to be feasible by Tout [Tou89], even if leather is thicker than paper. The following step consisted in analyzing the possibility of using glue instead of toner inside the laser printer. The dimension of the particles of toner is approximately 10-15 μm. Therefore for the experiment it was necessary to find a powdered glue with particles of comparable diameter, and which would be not sticky in this state. In addition toner can be magnetic and non-magnetic. If the transfer process could be shown to work with a laser printer which used non-magnetic toner it would have eliminated the problem of adding iron particle to the powdered glue. Since Bostic and Unex Dakota gave assurances that it was possible to obtain glue with the specified characteristics by grinding solid glue down to the desired dimensions, the selection process for a laser printer using non-magnetic toner could start.

From the literature and queries to the producers it has been possible to identify some companies which had done research on monocomponent insulating non-magnetic toner. The following commercially available printers using this toner have been identified:

- Ricoh PC Laser 6000;
- IBM Laserprinter 4019;
- OKI 40 OLI.

After some contacts with suppliers it was possible to acquire a second hand Ricoh PC Laser 6000 (the printer is not in production any more). The printer uses a mixing rod in order to charge the toner triboelectrically. A supply roller made of foam pushes toner against a roller which then must pass under a spring -loaded

metering blade before being transferred on the developer drum.

Before initiating the tests on the feasibility of the glue transfer it was necessary to empty completely the reservoir of the toner contained in it, and clean all the internal parts of the printer from the accumulated toner. Then the fusing system was disconnected in order to prevent the melting of the applied glue. This allowed a test of whether the glue would stay in place once applied.

Suppliers managed to provide two types of glue in powdered form, the Bostic HM10 produced by Bostic, and the Euremelt 2095 produced by Unex Dakota. Unfortunately a microscope analysis of these showed that they did not meet the required particle dimensions. The first one was mainly composed of particles whose average size was close to 150 μm, while the second one was a mixture of particles smaller than 200 μm. This posed a problem since the development system is designed to work with particles which are one order of magnitude smaller. A first trial with the Bostic HM10 showed almost no transfer. A second trial with Euremelt 2095 offered more promising results. It was decided to repeat the experiment with smaller toner particles. The process of separating smaller toner particles from the larger ones was carried out using a set of sieves obtained from the geotechnical engineering laboratory. A small sieve allowed the separation of particles which are larger than 65 μm. A smaller sieve able to filter particles down to 38 μm exists, but neither geotechnical engineering, nor geology had it. Eventually enough powdered glue to fill the cartridge with particle size inferior to 65 μm was produced.

All the experiments were carried out transferring the powdered glue onto paper. No trials with leather have been performed. As already reported, Tout has already shown the feasibility of transferring toner to leather with a laser printer, therefore proving that the process works on paper would lead to the conclusion that it is feasible on leather as well [Tou89]. However the most important reason that no experiments on leather were carried out is due to the fact that the laser printer employed was not designed to print on paper thicker than 200 μm. Trials with thicker paper had shown that transfer is possible, but it always jammed the printer at the level of the fusing roller. Since the leather is quite thick in comparison with paper it was not possible to insert it through the laser printer. Topis achieved printing

on leather by rebuilding part of the printer in order to allow thicker material to be passed through the rollers.

The results of the experiments show that glue transfer using an electrographic system is possible, but several problems were noted. First of all the glue is charged with the opposite sign in comparison to the toner powder, so the images were transferred in negative. This is not a problem for the deposition of the glue pattern, but interferes with the cleaning system of the laser printer which has to eliminate large quantities of glue from the photosensitive drum. In addition the transfer allowed only a thin layer of glue of be transferred, not sufficient for holding together components. The problem of the creation of toner of opposite sign is one yet to be addressed.

The results may look poor in comparison with the quality of the transfer which can be achieve with toner. However it is important to remember that in this case glue which has been just ground had been placed in the toner reservoir. A new research programme aimed at the development of a more suitable glue and the reengineering of the laser printer in order to allow the deposition of larger quantities of glue on the surface of the component will be required in order to implement a commercially valuable product.

5.5 Conclusions

In this chapter an example of the problems connected with the design of a device to be inserted in an automated manufacturing cell for the assembly of the upper of the shoes has been reported. This device is the only one which has not been developed among the industrial machines composing the second manufacturing cell presented in section 4.6. All the design activity has been based on the necessity of implementing a system which is capable of accepting components in random order and orientation, and of performing the appropriate operations on the component accurately in a short time.

The preliminary results reported here show that the approach of using a laser printer for transferring patterns of glue on the components is promising, but more

work needs to be done before a commercially valuable system is produced. In particular the re-engineering of the printer to allow leather to be printed is necessary. At the same time the development of a suitable form of powdered glue is essential. Unfortunately it has not been possible to assess the the wear of the photoreceptive drum due to the contact with the leather and Topis's research did not report any information on this subject. If problems of wear arise, the option of developing a glue transfer system based on the ionographic principle could perhaps be the best possible option.

The results obtained from the experiments carried on gave sufficient confidence in the technique to be able to include an automated electrographic gluing station as one of the manufacturing devices in the second of the workcells proposed in section 4.6.

Chapter 6

Computer Based Support System

6.1 Introduction

Modern production systems rely on information for every aspect of the production activity. Information must be efficiently stored, accessed and interpreted for production planning, and control as well as for managerial decision making. Computer based support systems are used within a production plant for allowing the organization and processing of data. Database systems store the static information which is used by the production planning system in order to organize the production. The interactions between these components allows the development a successful production strategy.

In the following sections an overview of the components of the manufacturing computer based support system is given. In section 6.2, the main features of manufacturing databases are presented. Section 6.3 gives an overview of the production activity control modules. Section 6.4 reviews some scheduling algorithms used in production control process and proposes a new heuristic scheduling approach.

6.2 Manufacturing Databases

In the field of distributed manufacturing systems access to a wide range of information is essential to guarantee correct operation of the plant. If insufficient data is available, proper support to the decision making process cannot be performed. Alternatively, a lot of unorganized data may also produced difficulties in the decision process.

Therefore it is highly desirable to provide distributed manufacturing systems with a properly designed information management system. This may be used for providing services in several of the manufacturing areas, such as: control, tooling, part programs, scheduling, as well as bills of materials and costing data.

There are two ways of storing the relevant information in a structured way inside a information management system; it is possible to use files, or a Data Base Management System (DBMS). Conventional files do not allow integrated processing, so independent programs have to use independent files. The data required by a program is bound to the application itself and its structure is declared within the application. This leads to the loss of flexibility and overall reliability of the system. DBMSs offer several advantages to file storage [Ran83]. DBMS software provides a representation of data that has logical (new fields can be added) and physical data independence (application programs are not affected by extensions to the structure). The same data can be accessed and shared by different applications, which do not need to be aware of the underlying storage structure. It can be organized as a collection of stored data for satisfying all the requirements of the application programs, and therefore no duplication of information is necessary. A common, controlled approach is used for adding, modifying or retrieving data. Built-in security and data integrity functions are often provided by DBMS packages. These advantages are paid for with a loss of speed since conventional files can be optimized for meeting the needs of application programs.

Database Systems are generally based on one of the four data models; hierarchical, network, relational and object oriented. Relational databases [Cod70] are a mature technology which have gained their popularity in shop floor applica-

tions due to their flexibility and relative ease of design and modification of their data structure. On the other hand the data model for network and hierarchical databases must be predefined, and if an application needs a new data structure, the DBMS must often be redefined and converted. Since the introduction of new products or processes is quite frequent in industrial automation, flexibility is an essential feature that a database must provide. Object oriented data bases offer even more flexibility than relational databases, but they do not yet represent a mature technology. Thus the relational model is believed to be the most suitable system for handling the data management aspects of the shop floor [BBB+91].

It is possible to divide the information within the manufacturing information system into two parts [KSN88] *static information* and *dynamic information.* The first one refers to the data which is not subject to frequent modifications. Due to its characteristics this data finds its best place inside a DBMS. On the other hand dynamic information concerns information which changes as the status of the production system changes. Since it is often updated and it is usually only of interest to local modules, it is usually stored as internal tables inside the application programs.

Static information included in a typical manufacturing database is concerned with data such as part design and program, routing, operation sequences, timing information, tooling information, and device information. Part design information contains detailed information on the part, and the programs store the data of the operations which have to be performed on it. Routing data defines the route, or the possible routes, that a part can take during its processing. This information is essential for efficient use of the machinery. Operation sequencing is concerned with the actual sequence of operations which a part has to undergo. Timing information is necessary to provide the system with the ability to generate a schedule for maximizing the use of the machinery present on the shop floor. Tooling data describes data on which tools are available, status, life,and overall use. Device information provides data on the devices present on the shopfloor, features description, manufacturer, productivity and related timing information.

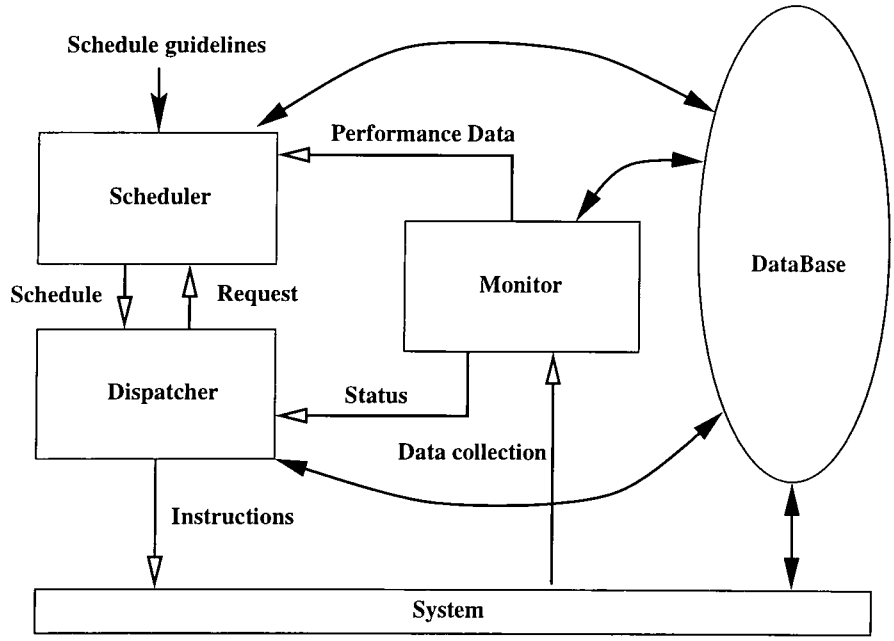


Figure 6.1: Production activity control modules and their interaction with the database and system

6.3 Production Activity Control

Production planning may be described as the process aimed at organizing material and component availability and at optimizing the use of the production capacity in a manufacturing plant [Wea88]. It is involved with all the activities from the acquisition of raw materials to the delivery of the complete products, and deals with management problems and their solutions. Its main functions are to provide a control strategy in accordance with the requirements stated by the management and to provide feedback on the status of the manufacturing process [FR88]. A production planning and control system may be divided into a layered structure. Each layer deals with a different aspect of the production process. The top layer is involved with the managerial objectives and reflects the strategy and the tactics of the company. Lower layers deal with inventory and production times and quantities. The Production Activity Control is located at the bottom level of this hierarchy. It has the role of an execution system accepting production orders from the higher levels, generating short term plans, and establishing priorities for all the job orders, so that they can be properly scheduled and manufactured by the production cells [BHS88]. Production plans received from higher levels are trans-

formed into control commands for the production process. At the same time it controls and evaluates the production activities of the manufacturing organization, and produces useful feedback information to the higher level planning functions.

The building blocks of the Production Activity Control system and their interaction with the system and the database are shown in figure 6.1. They are the *scheduler, dispatcher* and *the monitor* [BBB+91]. The scheduler develops a production plan over a specific period of time. The dispatcher uses the schedule for issuing the appropriate commands to the production system. Since the production environment is dynamic the monitor holds information related to the current status of the system which is used by the dispatcher to generate the control commands. Sometime other two additional modules are included in the Production Activity Control, the *mover* and the *producer* [BBB+91]. They are under direct supervision of the dispatcher and are usually involved with control at workstation level and handling of material between workstations. The following sections will give a more accurate description of the tasks performed by the modules included in a Production Activity Control system.

6.3.1 The Scheduler

The task of the scheduler consists in accepting the production requirements from the higher levels of the production system and generating a plan which allows the manufacturing of the required goods optimizing some of the parameters (e.g. production time, machine utilization etc.). Scheduling is a complex activity which is dependent on several factors included the topology of the system, the complexity of the involved operations, and the predictability of the involved processing times [McM93]. Therefore simple and well designed manufacturing processes are easier to schedule that complex ones. A brief review of the large number of scheduling algorithms which have been developed in the past is presented in section 6.4. The development of the actual production schedule can be based on algorithms or simulation packages or a combination of the two [Car88, BBBB91]. The generation of the schedule is a process composed of three main activities [BBB+91]. First it

is checked that the production requirements generated by the higher level can be implemented in a feasible schedule. If not the appropriate warnings are reported to the higher levels. Then the actual schedule is generated taking in account of the constraints of the system and with the target of minimizing or maximizing some of the parameters. Finally the schedule is passed to the dispatcher. If the production requirement are incrementally communicated to the scheduler its activity is described as real-time scheduling since the schedule is updated with the passage of the time in order to accommodate the new production requirements.

6.3.2 The Dispatcher

The dispatcher can be seen as the real-time scheduler which formulates the final commands for the production system. It generates the orders on the basis of the schedule received from the scheduler, from the information related to how the processes are performed and from the snapshot of the current status of the system stored in the monitor. The activity of the dispatcher is essential since the manufacturing environment is not deterministic, and delays, exceptional or abnormal situations may arise. Its task is to ensure that the schedule is followed as closely as possible using different possible algorithms [McM93]. In the case of anomalies in the production it has to take the appropriate actions so as to minimize the effects of the disturbance to the schedule, resequencing jobs and reallocating resources. If the schedule cannot be achieved, appropriate actions involving a possible rescheduling from the scheduler should be taken.

6.3.3 The Monitor

The monitor collects information about the activities performed inside the production cell and makes them available to the scheduler and the dispatcher. For this reason it can be seen as a translator from *data* into *information* [McM93]. It captures data on the events which take place on the plant floor, such as process

timings, machine status, material availability, and machine downtimes. Then the monitor analyzes this data in order to make it available in a proper and meaningful format to the other components of the production activity control and the higher levels of the planning hierarchy. Data which may be gathered during these operations includes monitoring of the work in progress and the status of the resources, consumption rates of materials, and quality related data [BBB+91]. It offers data which can be used in the decision support activity by the scheduler and the dispatcher; this data includes the current status of all the devices, job in progress related data and material availability. Some of this information may be passed to the higher levels of the production hierarchy for assessing the productivity of the cell and monitoring the plant at a higher level.

6.4 Scheduling Techniques

The scheduling problem is involved with setting the timetable for the processing of jobs on to machines so that a given measure of performance achieves its optimal value [McM93]. Many researchers have generated approaches for solving these problems which have an immediate application in many the fields of manufacturing. The large number of works in this area are related with the large number of variables involved in the process to schedule, the parameters to optimize, and the internal combinatorial explosiveness of the problem. In [ML93] a comprehensive review on optimization and heuristic methods in production scheduling is presented. Approaches for solving scheduling problems using knowledge-based approaches are described in [NS91], while [HR91] presents a review of real-time scheduling techniques.

Scheduling algorithms are usually designed to solve special problems involving constraints on jobs and scheduling objectives [ML93]. The flow pattern may be the same for all the jobs (*flow shop*), or each job may have its own individual flow pattern (*job shop*), or no specified flow pattern may exist (*openshop*). Parallel or duplicated machines may be present. Within these environments it is possible to achieve a schedule which tries to optimize an objective. This may consist in *com-*

pletion time, *flow time*, *tardiness* or *waiting time* of the jobs.

The workcell proposed in chapter 4 for the assembly of the uppers has the constraint that all the jobs have to follow the same path, and that once a job is started it must be processed until completion without any interruption either on or between machines. A comprehensive review on the state of the art in such an area of scheduling is presented in [HS96]. Since it is possible that in the future new machines will be included in the proposed workcell, the problem of finding a scheduling algorithm for the m machine flowshop was required. In particular the objective of minimizing the total flow time (Makespan), the sum of all the completion times of the jobs, was addressed. The scheduling problem associated with the proposed workcell can be defined as $F_m||no-wait||\sum C_j$, where F_m indicates a flowshop problem with m devices, and C_j is the completion time of the job j. It was decided to minimize the total flow-time since this criteria minimizes the sum of completion times, and therefore it aims to finish each job as soon as possible. This is an appropriate criteria in the case of the proposed cell, since minimizing the average times it takes to complete a job is a sensible target in a situation where new jobs are constantly added to the job list. There are only a few available algorithms which have so far been proposed for solving this problem. Van Deman and Baker [DB74] proposed a branch and bound approach to find the optimal solution proposing a set of procedures for generating lower bounds on optimal values. Some theoretical work on the problem has been developed by Panwalkar and Woollam [PW80] and Adiri and Pohoryles [AP82]. They describe some methods which aim to find optimal schedules in particular situations. Several heuristic algorithms have been proposed by Rajendran in collaboration with others [RC90, GR93, Raj94]. These algorithms when compared with the ones developed by Bonney and Grundy [BG76] and King and Spachis [KS80], produce schedules which are closer to the optimal ones. However these algorithms are quite complex to implement and they are mainly based on the evaluation of the partial flow times of the generated partial schedules.

The research carried out in this area led to the implementation of a new heuristic approach for the generation of a schedule whose performance is comparable to some

of the algorithms proposed by Rajendran. The main strengths of this approach are its simplicity, and implementation ease. The foundations of the algorithm are given in the next section.

6.4.1 The proposed algorithm

It is possible to observe that several of the algorithms which are used for the computation of schedules are based on branch and bound approaches which analyze partial schedules and usually expand only the one which maximizes (or minimizes) the desired objective. The hypothesis which is at the base of the heuristic algorithm proposed here is related to the fact that is it much simpler to evaluate just pairs of jobs than scheduling them to evaluate (partial) combinations of all jobs.

The algorithm proposed is based on the work done by Chan and Bedworth [CB90] and it has been adapted for the *no-wait* flowshop case. The algorithm has been designed to solve the problem of minimizing the mean flowtime in the n-job/m-machine case in static and dynamic environments. The extension of the methodology for the *no-wait* flowshop environment has proved to be possible maintaining the same approach.

The assumptions made from Chan and Bedworth had to be modified in order to make the algorithm compatible with the no-blocking case. They are:

1. job operation times are deterministic and known in advance;
2. an operation once started on a machine cannot be interrupted on a machine before completion (no pre-emption);
3. jobs consist of a strictly ordered sequence of operations;
4. the same job sequence is assumed for the same part family, but machine skipping is allowed;
5. once a job is started, it must be processed until completion without any interruption either on or between machines.

These assumptions are compatible with the processing activity performed by the proposed workcell.

The following notation has been used:

n	number of jobs to be scheduled
m	number of machines in the flow shop
t_{ij}	processing time for the ith job on the jth machine
$F_{m(ij)}$	total flowtime for job i followed by job j on m machines
$F^*_{m(ij)}$	simplified flowtime for job i followed by job j on m machines

Chan and Bedworth develop the formula which is the base of their heuristic starting from the fact that in a two-job, two-machine environment where job i precedes job j, there are two possible flowtime sequences as shown in figure 6.2. From the part A of the figure, if $t_{j1} > t_{i2}$, it is possible to see that the flowtime for job i is equal to $(t_{i1} + t_{i2})$. Similarly the flowtime for job j is $(t_{i1} + t_{j1} + t_{j2})$. The total flowtime is the sun of the two flowtimes:

$$F_{2(ij)} = 2t_{i1} + t_{i2} + t_{j1} + t_{j2} \tag{6.1}$$

Instead if $t_{j1} < t_{i2}$, as in the part B of the figure, the total flowtime is equal to:

$$F_{2(ij)} = 2t_{i1} + 2t_{i2} + t_{j2} \tag{6.2}$$

However in the case of a *no-wait* flowshop case the Gantt diagrams showed in figure 6.2 are modified as in figure 6.3. It is possible to see that the equations 6.1 and 6.2 are still valid and therefore the approach developed by Chan and Bedworth can be immediately extended to the *no-wait* flowshop case. The detailed mathematical development of the formula which is the base of the proposed scheduling heuristic is reported in appendix B.

Job sequences can be generated using the following algorithm;

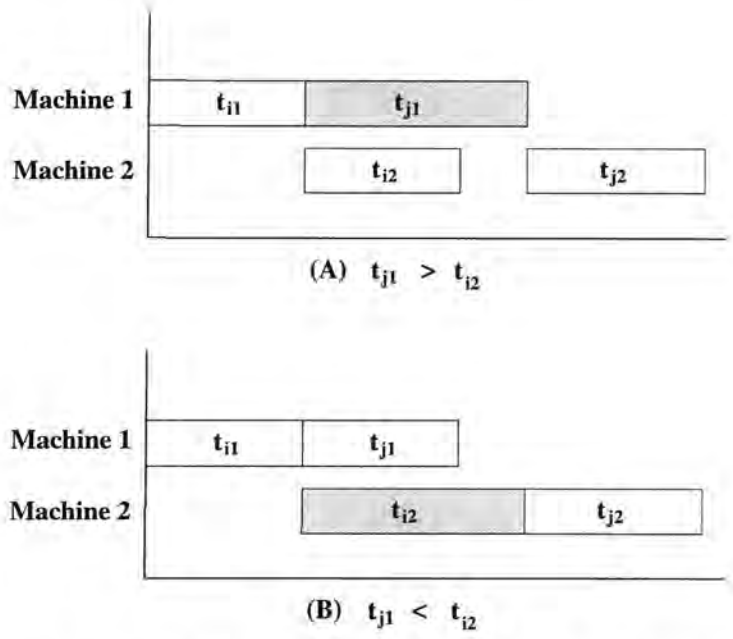


Figure 6.2: Gantt charts for the two machine problem in the blocking flowshop case

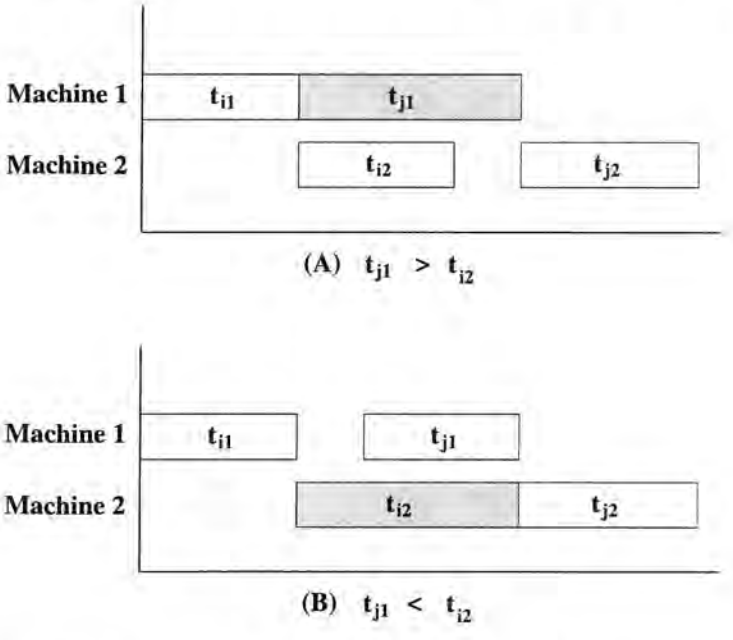


Figure 6.3: Gantt charts for the two machine problem in the *no-wait* flowshop case

1. Using the processing times compute temporary, simplified flowtimes for each pair of jobs using equation B.9;

2. compare each time of temporary flowtimes (e.g. $F^*_{m(pq)}$ and $F^*_{m(qp)}$) and select the smallest and mark the starting job of the pair;

3. perform step (2) for all the pairs of flowtimes;

	Machines			
Job	1	2	3	4
1	15	24	11	18
2	22	5	19	13
3	16	20	7	16
4	12	25	16	17
5	15	15	18	25

Table 6.1: Processing times for a five jobs four machines problem.

Sequence pair	F^* values	Optimum pair
1-2;2-1	119;118	2*-1
1-3;3-1	118;114	3*-1
1-4;4-1	130;125	4*-1
1-5;5-1	122;121	5*-1
2-3;3-2	111;105	3*-2
2-4;4-2	121;125	2*-4
2-5;5-2	116;121	2*-5
3-4;4-3	120;123	3*-4
3-5;5-3	112;121	3*-5
4-5;5-4	124;121	5*-4

Table 6.2: Pair of sequences, their simplified flowtimes and optimum pairs

4. When all the pairs have been evaluated count the number of marks by each job, and sequence them in decreasing number of marks. If more jobs have equal number of associated marks sequence them in decreasing total processing time order.

The decision to schedule jobs in decreasing total processing time order when they have an equal number of marks derives from the heuristics proposed by Rajendran [Raj94], where he assumes that in order to minimize the makespan the last job should have a short processing time. This is true for partial sequences as well.

6.4.2 Scheduling example

Consider a five-job four-machine no-wait flowshop example. The processing

times are reported in table 6.1.

It is necessary to calculate all the possible combinations of the partial flowtimes: the results of the 20 simplified flowtimes are reported in table 6.2. The temporary flowtimes for the sequences 1-2 and 2-1 will be calculated to exemplify the computational process.

For the sequence 1-2 using B.9;

$$F^*_{5(12)} = 2t_{11} + (t_{12} + t_{13}) + R^*_{4(12)} \tag{6.3}$$

The last term can be calculated as follows:

$$R^*_{4(12)} = max(t_{23} + max(t_{22} + max(t_{21}, t_{12}), t_{12} + t_{13}), t_{12} + t13 + t_{14})$$

$$= max(19 + max(5 + max(22, 24), 24 + 11), 24 + 11 + 18)$$

$$= max(19 + max(5 + 24, 35), 53)$$

$$= max(19 + 35, 53)$$

$$= max(54, 53) = 54$$

and therefore equation 6.3 is equal to:

$$= 2(15) + (24 + 11) + 54 = 119$$

Similarly for sequence 2-1 we have:

$$F^*_{5(21)} = 2t_{21} + (t_{22} + t_{23}) + R^*_{4(21)} \tag{6.4}$$

where the last term is calculated using:

$$R^*_{4(21)} = max(t_{13} + max(t_{12} + max(t_{11}, t_{22}), t_{22} + t_{23}), t_{22} + t23 + t_{24})$$

$$= max(11 + max(24 + max(15, 5), 5 + 19), 5 + 19 + 13)$$

$$= max(11 + max(24 + 15, 14), 37)$$

$$= max(11 + 39, 37) = 50$$

and therefore equation 6.4 is equal to:

$$= 2(22) + (5 + 19) + 50 = 118$$

Since $F^*_{5(21)}$ is smaller than $F^*_{5(12)}$ job 2 is marked. The pair evaluation report is given in the third row of table 6.2. The sum of the marks of the sequence is; job 1= 0, job 2 = 3, job 3 = 4, job 4 = 1, job 5 =2. After ordering the jobs in decreasing mark order the schedule 3-2-5-4-1 is generated whose flowtime 521 is optimal.

The heuristic approach allowed us to consider only 10 combinations of jobs instead of the 120 (5!) required for a complete search of the space of the solutions.

6.4.3 Computational experience

The proposed heuristic algorithm and the best algorithm proposed by Rajendran [RC90] have been implemented in Java [GM95], as it was selected for the development of the system (the language is described in chapter 9). Over 600 problems, with number of jobs ranging from 5 to 9, and number of machines ranging from 3 to 25, have been generated and solved by the two implemented algorithms as well as optimally doing an exhaustive search among all the possible schedules. The processing time of the jobs were randomly generated from a rectangular distribution ranging from 1 to 99. The two heuristics were evaluated on the basis of the deviation from the optimal solution, calculated doing an exhaustive search in the solution space, using the formula:

$$\frac{(flowtime_{heur} - flowtime_{opt}) * 100}{flowtime_{opt}}$$

No of Jobs	No of Machines	No of Problem	Proposed Heuristic		Rajendran's Heuristic	
			mean	stdv	mean	stdv
	5	30	1.8974	2.2508	0.5598	0.9610
	10	30	2.5422	3.3237	0.2527	0.7112
5	15	30	2.0531	2.6756	0.3723	0.8758
	20	30	1.0671	1.3019	0.1052	0.2941
	25	30	1.8812	2.0223	0.2677	0.5881
	5	30	3.9568	3.9397	0.8007	1.1791
	10	30	2.8787	2.9374	0.2281	0.4342
6	15	30	2.2136	2.4889	0.1586	0.4358
	20	30	1.7303	1.8211	0.2791	0.5435
	25	30	1.8497	1.9204	0.4766	0.7667
	5	30	5.0019	3.7175	1.0133	1.8249
	10	30	3.6980	3.0086	0.4387	0.7669
7	15	30	4.3286	2.7606	0.5963	1.2191
	20	30	3.8280	2.7875	0.7038	0.9763
	25	30	3.1962	2.4739	0.7106	0.8782
	3	20	7.5960	4.2466	1.4090	2.2503
	6	20	6.6500	4.5565	1.0975	1.6325
8	9	20	6.5085	3.4767	0.7504	0.7366
	12	20	5.2020	3.3753	1.2042	1.2862
	15	20	3.7619	2.5321	1.1963	1.5525
	3	20	7.0923	5.3553	1.5945	2.4075
	6	20	7.1540	4.4810	1.7364	1.6179
9	9	20	5.4558	4.0878	1.1558	1.1242
	12	20	5.4769	2.1512	1.2956	1.0887
	15	20	5.3129	3.6578	0.5622	0.6365

Table 6.3: Mean and standard deviation of the heuristic solution against the optimal solution in the case of the proposed heuristic and those of Rajendran

The mean of the deviation from the optimal solution and the variance are reported in table 6.3.

Since the results reported by the heuristic proposed by Rajendran were in accordance with the results reported in his paper it has been possible to compare the proposed algorithm with other heuristic algorithms implemented by Rajendran and used in his article for comparison, namely the heuristic proposed by Bonney and Gundry [BG76] and the one presented by King and Spachis [KS80]. These

No of Jobs	No of Machines	No of Problem	Proposed Heuristic	Rajendran's Heuristic	Bonny's Heuristic	King's Heuristic
			Mean absolute percentage error			
	5	30	1.8974	0.5598	4.3012	22.2497
	10	30	2.5422	0.2527	2.9883	16.4763
5	15	30	2.0531	0.3723	2.0262	12.9883
	20	30	1.0671	0.1052	3.0443	14.1733
	25	30	1.8812	0.2677	2.3930	11.8693
	5	30	3.9568	0.8007	5.8617	24.8628
	10	30	2.8787	0.2281	3.5217	18.6120
6	15	30	2.2136	0.1586	2.8151	13.4577
	20	30	1.7303	0.2791	2.8823	15.1923
	25	30	1.8497	0.4766	2.8857	13.2613
	5	30	5.0019	1.0133	7.0663	25.9993
	10	30	3.6980	0.4387	3.3460	18.9526
7	15	30	4.3286	0.5963	3.6213	15.8137
	20	30	3.8280	0.7038	4.7290	15.9730
	25	30	3.1962	0.7106	3.4060	14.0723
	3	20	7.5960	1.4090	7.1225	24.8465
	6	20	6.6500	1.0975	7.0180	18.8380
8	9	20	6.5085	0.7504	4.7250	19.0430
	12	20	5.2020	1.2042	4.0305	13.3915
	15	20	3.7619	1.1963	5.4700	13.0835
	3	20	7.0923	1.5945	9.9805	30.2420
	6	20	7.1540	1.7364	7.6225	23.1771
9	9	20	5.4558	1.1558	5.4215	21.8795
	12	20	5.4769	1.2956	5.1475	16.2595
	15	20	5.3129	0.5622	6.3795	15.5965

Table 6.4: Mean of the heuristic solutions against the optimal solution of the proposed, Rajendran's, Bonny and Grundy, and King and Spachis's heuristics.

results are reported in table 6.4.

The proposed heuristic does not perform as well as the best one proposed by Rajendran, but it usually performs better than the one proposed by Bonney and Gundry and generates better solutions than the one proposed by King and Spachis. The reason for the better performance of the heuristic of Rajendran is that it performs a selection of which is the best path doing a search on partial job sequences. The proposed algorithm does not consider partial sequences, but derives the solu-

No of Jobs	No of Machines	Proposed Heuristic			Rajendran's Heuristic	
		mean (ms) [a]	std dev	mean (ms) [b]	mean (ms)	std dev
5	5	4.233	0.869	6.233	9.966	0.884
	10	6.133	0.571	14.466	10.500	0.861
	15	8.433	0.678	27.200	11.166	0.647
	20	11.000	0.789	44.733	11.666	0.884
	25	12.966	0.490	64.566	12.400	0.932
6	5	5.366	0.556	8.366	13.700	0.749
	10	8.966	0.718	21.466	14.466	1.008
	15	12.233	0.430	39.133	14.866	0.507
	20	15.900	0.711	64.966	15.666	0.922
	25	19.766	0.720	96.266	16.233	0.626
7	5	7.100	0.305	11.533	18.500	0.937
	10	11.966	0.868	29.366	19.466	0.819
	15	17.266	1.436	56.866	20.133	1.041
	20	22.366	1.684	90.100	21.133	1.332
	25	27.666	1.625	134.433	21.466	1.507
8	3	7.333	1.073	10.800	26.000	1.982
	6	10.600	0.855	20.366	26.566	1.590
	9	14.533	0.628	40.966	26.733	0.583
	12	18.566	0.935	60.866	29.633	1.637
	15	23.966	1.943	86.333	29.133	1.125
9	3	8.633	0.927	11.233	32.933	1.014
	6	13.800	0.961	27.866	33.833	0.949
	9	18.866	1.074	48.833	36.433	1.863
	12	23.833	1.341	77.433	35.600	1.823
	15	28.800	1.095	111.866	35.500	1.137

[a] non-recursive implementation
[b] recursive implementation

Table 6.5: Evaluation of the heuristic solutions against the CPU time requirement. The results show mean and standard deviation of the time required over 30 runs. The timing of the proposed heuristic shows the results related to the non-recursive and recursive implementation of the algorithm.

tion from comparing pairs of sequences of two jobs.

Finally an evaluation of the CPU time requirements of the proposed algorithm and that of Rajendran has been carried out. These tests were performed on a Pentium 75 MHz running Linux 2.0.0 and the mean time and standard deviation for runs of 30 problems are shown in table 6.5. It is possible to notice that the processing time of the heuristic proposed by Rajendran is independent of the number of

machines. This is not true for the proposed algorithm where the computational requirements increase with the number of machines. During the tests the importance of the actual implementation of the algorithm was noted. At first the function for calculating the $R^*_{m(ij)}$ was implemented in a recursive way. This generated a poor performance since the nesting of calls to the method increased with the number of machines. When the same method was rewritten using a non-recursive approach better results were achieved. The proposed algorithm performs better than that of Rajendran for a small number of machines, but its performance decreases as the number increases.

Summarizing, the proposed algorithm produces results comparable with those proposed by other researchers. It is simple to use, simple to understand, and generates consistent meaningful schedules which identify it as a good heuristic algorithm according to the criteria stated by Melnyk and Carter [MC87].

6.5 Conclusion

The control of the production activity is an important and complex task in modern manufacturing plants. A brief overview of the components involved has been presented in this chapter. The role of the database system in providing vital information to the whole production system has been emphasized. Components and device features are stored as entries in the tables of the database. This information is essential to the Production Activity Control system whose task is to elaborate the correct actions to optimize the production capacity of the controlled cell. It is constituted by three interacting modules, the monitor, the scheduler and the dispatcher. The information held in the database, in conjunction with that related to the actual status of the plant, is elaborated by the Production Activity control system in order to generate and execute a production schedule in accordance with the guidelines given by the higher levels of the production hierarchy.

Since the second of the manufacturing cells proposed in section 4.6 has the feature that once a job is started it must be processed until completion without any interruption either on or between machine, a suitable scheduling algorithm

had to be used. The search of the appropriate scheduling heuristic has lead to the development of a new scheduling approach based on an algorithm developed to resolve a similar problem. The algorithm has the advantage that it is simple to understand and to implement, and its performance has proved to be comparable with more complex algorithms. In addition the criteria used by the scheduling algorithm of minimizing the makespan, and therefore aiming to finish each job as soon as possible, is believed to be appropriate for the manufacturing problem.

Chapter 7

System Design

7.1 Introduction

The design stage of an application covers all those activities from the identification of the requirements of the system to its implementation. For this reason the design stage represents the bridge which connects the specification phase to the development stage.

The design stage is concerned with the definition of the modalities which allow the passage from "what" has to be implemented to "how" it has to be implemented.

The necessity for a system design process comes from the fact that many systems are inherently complex. The first aim of any design technique is to overcome such complexity: in order to reduce any system to a tractable one, it is necessary to divide it into subsystems. In this way it is possible to decompose a large system into a set of smaller and simpler ones.

The aim of this exercise is to obtain two fundamental results:

- the "sum" of the complexity of the single parts must be lower than the total complexity of the original system;
- the subsystems can be developed and implemented independently by different people who can work on them at the same time thus reducing the overall production time.

The design stage is critical since at this point all the key decisions for the development of the system are taken, and these decisions often generate constraints which may be binding during the activities of system development.

In the specific area of manufacturing applications the design stage has to produce a system which is kept as simple as possible. Dividing the system into subsystems, identifying every input and output, and determining the appropriate controls, are necessary steps for a successful implementation. In addition the development approach must produce a system which could be able to cope with future modifications, since new products may require the partial restructuring of the plant. In addition it must be able accurately to model the system in order to identify all the requirements, since the modification of computer systems once installed is usually an extremely expensive task [MM88].

Several books focused on design of manufacturing systems, such as [McM93] and [Ran83], tend to provide a general overview on the components which are present in an industrial environment without proposing a proper design methodology. However some design techniques especially addressed to the development of manufacturing system exists: GRAI [DVDR87], and the design methodology proposed by Wu [Wu92] are two examples. Unfortunately GRAI is specially addressed to the design of the managerial system of a manufacturing plant, and therefore it is not suitable for the development of a communication and control system. The design methodology proposed by Wu tries to embrace all the aspects of the development of of a manufacturing system and therefore it appears to be too complex for the design of the proposed plant support system.

Due to the fact that it has not been possible to identify a design approach targeted to the development of manufacturing system which could be used for modelling a communication and control system for the footwear industry, a more general methodology had to be selected. However the lack of a targeted design methodology did not constitute a problem, since the scope of the project consisted in the implementation of a software support system. Therefore normal approaches used for software development could have served the scope. Two well accepted methodologies SSADM [Ash88], and SADT [MM88] have been analyzed in turn,

but they have both been rejected as unsuitable. SSADM provides a very good support for the definition of the requirement of the system and its specification, but it does not give clear guidance on how to move from the design to the actual code. SADT does not provide a good support for understanding timing relations among the components of the system, essential feature in a communication system.

An approach which allowed the analysis of the different features of the system as well as providing examples of how to move from the design stage to the coding was found in the *Object Modeling Technique, OMT* [RBP+91].

This chapter provides an overview of the design stages which preceded the implementation of the shop floor system. In section 7.2 a description of the design methodology selected is given. Then in section 7.3 OMT is applied for defining the proposed manufacturing support system.

7.2 Object Modeling Technique

The selection of the most appropriate technique to be used for the design of the manufacturing cell has been mainly conditioned by the complexity of the system. An integrated support system for controlling footwear workcells presents particular features and complex functions which are related with the exchange of messages and the coordination and interaction of the single components of the system. In addition the search for a suitable technique for the design of the system was conditioned by the fact that the model would have to capture the "dynamic" aspect of the system as well as the "static" one. On the other hand the implementation of the software without a proper design stage would have undermined the quality of the final product. All these elements have suggested OMT, as the most suitable method to be employed during the design stage. This technique proposes a general framework which can be applied to any area where system design is required. The selection of OMT is due to the fact that it allows the design of an application in all its details, moving in steps from a high level of abstraction towards less abstraction and leading to the full description of all the most important aspects of system. Moreover the OMT technique uses an object oriented approach which

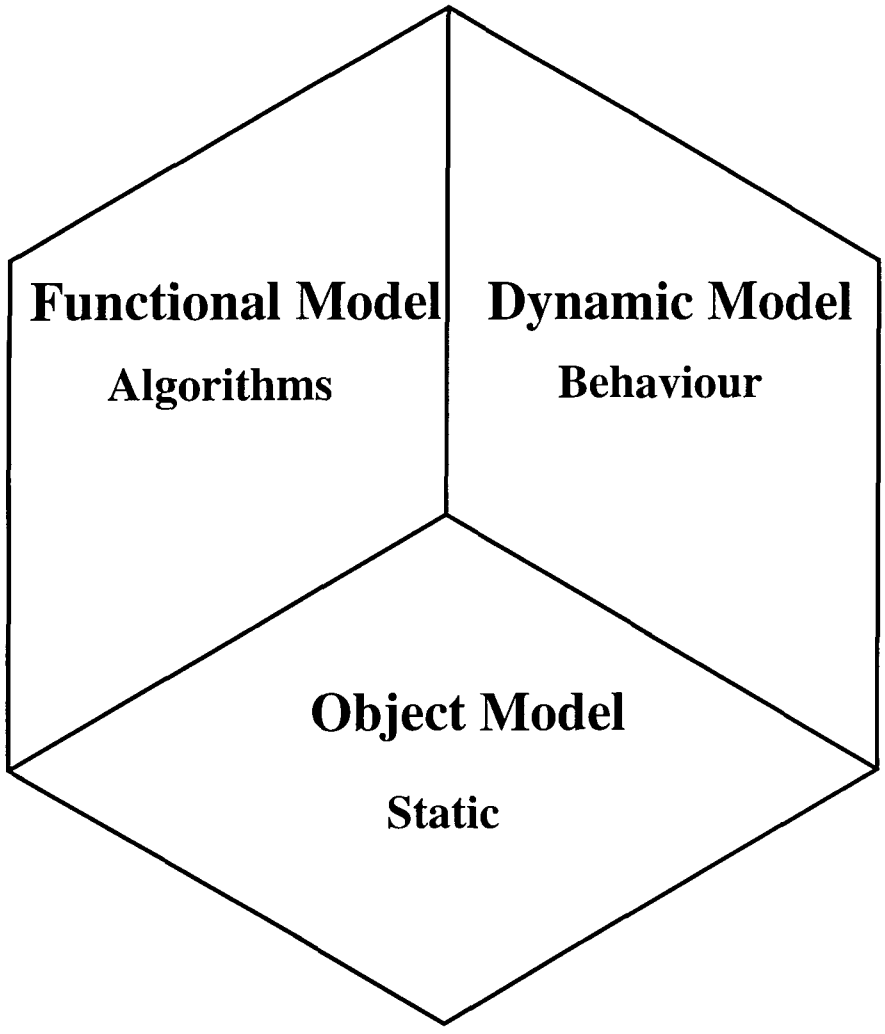


Figure 7.1: OMT models

means that the system, the manufacturing communication and control system in our case, is seen as a set of objects (the devices, the database, the control system) with their own attributes, properties and behaviours. The events used in the OMT technique are easily mapped into the manufacturing cell; for example the introduction of new components to be processed, or the exchange of messages can be represented as events. Finally the OMT technique allows the implementation of complex algorithms, such as the ones resident in the control system, covering all the aspects of the design of an application.

Object oriented modelling can be interpreted as a way for abstracting a problem using elements coming from the "real world" instead of concepts coming from the "computer world". The term *object oriented* means, *prima face*, that the software

is organized as a collection of objects which embody both the data structures and their behaviour. This is in contrast with conventional programming where data structures and functions are only partially connected. An object oriented approach presents some peculiar characteristics which can be summarized in:

- identity;
- classification;
- polymorphism;
- inheritance.

Identity represents the fact that data is represented using discrete, distinguishable entities called *objects*. Examples of objects are the paragraph of a document, a piece in the game of the chess, a computer in a network. The object may have a physical structure or be only conceptual. The identity is expressed due to the fact that different objects, even if represented using the same attributes, such as name and dimensions, have their own inherent identity.

Classification means that objects that have the same data structure (attributes) and behaviour (operations) are grouped in a single class. Paragraphs, chess pieces, computers are examples of classes. Which classes are relevant and how they are structured depends on the application, since a class is designed to describe only properties which are relevant for the application ignoring all the rest. A class specifies a possibly infinite set of objects. A single object is said to be an *instance* of its class.

Polymorphism represents the fact that the same operation may behave in a different way if applied on different classes. For example the moving operation will produce a different outcome if applied to a paragraph in a document and to a piece in the game of chess. An operation is an action or a transformation that an object performs or is subject to. A specific implementation of an operation by a certain class is called a *method*.

Inheritance represents the possibility that classes have to share attributes and operations based on a hierarchical relationship. A class can be defined in a broad

way and then refined into subclasses. Each subclass inherits, and therefore incorporates, all the properties defined in its *superclass* and adds its specific properties. The possibility of grouping together classes in a hierarchical fashion can reduce repetition in data structures and programs.

In addition object oriented technology provides the possibility of *data abstraction* and *encapsulation.* The first one allows the developer to focus on inherent aspects of an entity and therefore on what the object is and does, before deciding how to implement it. The second, called sometimes *information hiding* allows us to separate the external properties of an object, which are accessible by other objects, from the internal implementation characteristics, which are hidden to the external world.

The development of an application using a object oriented technique may be considered as a conceptual process independent from the programming language employed. Object oriented design is therefore a conceptual framework and not a programming technique. One of the great benefits of using an object oriented technique for the design of a system consists in the possibility which is granted to developers, programmers and users to express in a clear and simple way abstract concepts, and communicate them to each other. It is important to point out that the introduction of this methodology usually does not reduce the design time if compared with other design techniques. However its intrinsic clear and simple structure shows its advantages when it is necessary to perform revisions or modification of the system. Finally it allows the development of more robust applications, reducing the possibility of unexpected behaviour, since the design approach is extremely structured.

The OMT design approach permits the definition of the system starting from a high level of abstraction through a series of steps which specify elements connected with the implementation of the application.

The OMT methodology uses three models of typology for the complete design of a system or application: the *object* model, the *dynamic* model, and the *functional* model (see figure 7.1).

The object model describes the static structure of the objects in a system and

their relationships within the system. It is represented through an *object diagram* which is a graph where the nodes represent the object classes and the arcs represent the relationships among classes. In the case of the design of the manufacturing cell, the objects identified by the object model at the top level can be identified with the different elements present in the system, such as the devices and the database.

The dynamic model describes the aspects of the system which are subject to modification during the execution of the application. It allows us to specify and implement the control aspects of the system or the implementation. The dynamic model is composed of state diagrams which are graphs where the nodes represent states of the system and the arcs are the transitions between states which are fired by specific events. In the design of the manufacturing cell different sources of events can be identified, and several of them are related with message exchange between devices.

The functional model describes the transformation of the data inside the application. It is based on data flow diagrams which represent the computations which are carried out by the application.They are graphs whose nodes represent processes and the arcs represent the flux of the data flow. The functional model is used for the design of complex algorithms and it in this project it is particularly suited for the development of the scheduling algorithm.

The three models described in the OMT technique are *orthogonal* in the complete description of an application and they are tightly connected to each other. However the fundamental one is the Object model, since first it is necessary to describe what is subject to transformation (the object) before describing when or how it changes.

One of the main strengths of the approach is the fact that a discussion of the specific details for implementing a system using object-oriented languages, non-object oriented languages, and databases management systems, starting from the design is carried on. This methodology is extremely valuable since it proposes a framework for translating the models created during the design stage into a working application.

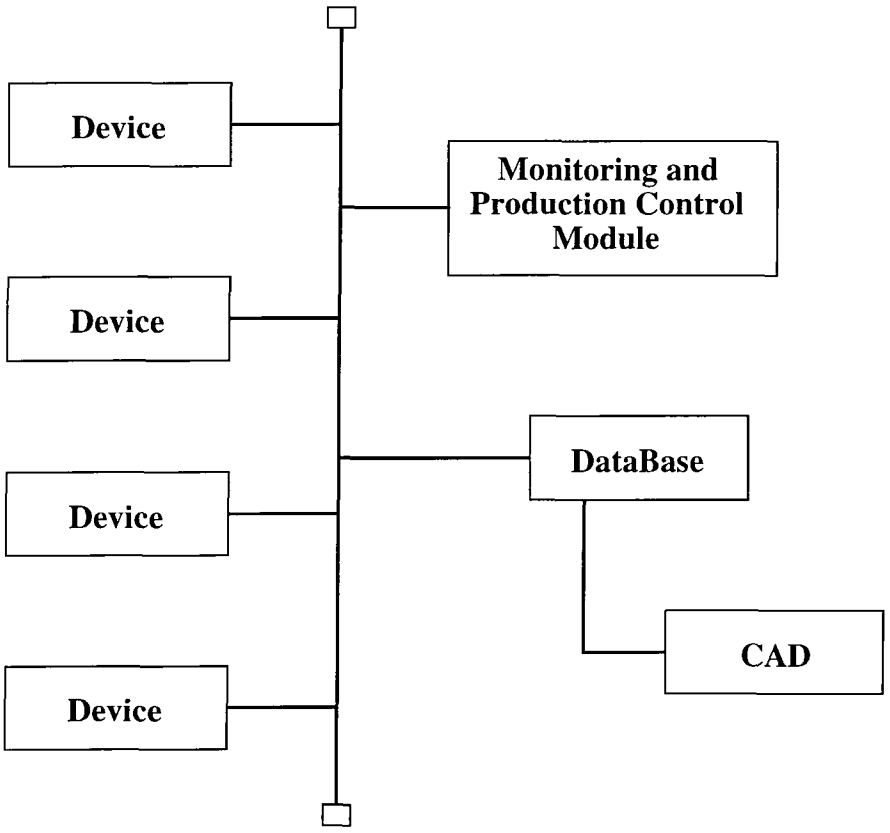


Figure 7.2: Schematic diagram of an interconnected manufacturing system for the footwear industry

7.3 System Analysis

The aim of the design stage is to provide the structure for a general production support system for a networked manufacturing cell which could be used to automate any stage of the production chain. The particular manufacturing cells described in chapter 4 will be used as a special case to validate the design. A schematic diagram of an interconnected manufacturing system for the footwear industry is shown in figure 7.2. In the following subsections the Object, Dynamic and Functional Models of a manufacturing cell based on the one shown in figure will be developed.

7.3.1 Object Model

During the design of the system, due to the large number of object classes present in the object model, it was necessary to group them in *modules*. A module is a

logical construct for grouping classes, associations and generalizations. The convention of indicating a module with a small triangle placed in the top right corner of the box has been used in all the object diagrams. Modules are then expanded during the design stages. At this stage of the project only the main attributes associated to the objects will be discussed.

It is possible to extract the relevant objects from the requirements and from the knowledge of manufacturing and networking issues. Since, as mentioned, the number of object classes necessary to give a full description of the system is large, at the top level several correlated classes were collapsed into modules. From the analysis it is possible to identify the following object classes and modules (in bold font) necessary to describe the system:

Device	**Interface**	**Production Control**
CAD	**DataBase**	Component
Device Data	**Component Data**	Processing Time
DBMS		

The *Devices* represent all the manufacturing devices which need to communicate with the external world. The Devices communicate with the Production Control and the DataBase through the Interface. The object class Device is a module. Every networked entity present in the system has an *Interface* which encodes outgoing messages and decodes incoming ones setting the rules for inter-device communication. The *Production Control* represents the set of systems which coordinate the activity of the production cell. In order to achieve its task it needs to know the *Components* which have to be processed by the system. The Production Control communicates with the Devices and the DataBase through the Interface. The *CAD* is the source of the data used by the industrial devices for processing the components. It accesses the DataBase for loading and downloading information related to the components. The *DataBase* is the repository of all the information about the devices present in the cell and the components manufactured. It is managed by the *DBMS*. It communicates with the Production Control and the Devices through the Interface. The *Device Data* is stored in the database and holds information relevant to the devices. Each Device has associated a Device

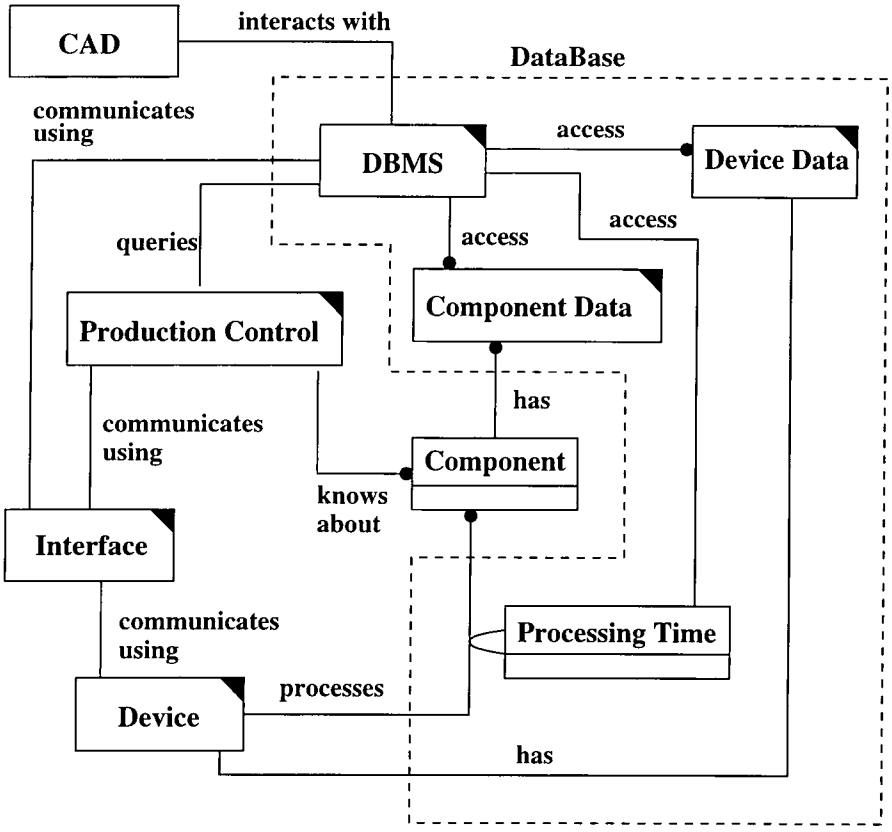


Figure 7.3: Top level Object Model for the manufacturing cell

Data object in the DataBase. The *Component Data* and the *Processing Time* are stored in the DataBase as well and hold information relevant to the components processed by the system.

The top level Object model of the manufacturing cell is displayed in figure 7.3. It is possible to identify the interconnections among the object classes present in the system. The dotted line which groups some of the components defines the boundary of the DataBase module. Since it has several associations with the other elements represented in the system, it has been expanded at top level for the sake of clarity. It is seen that all the components of the system need to use the Interface module for communicating. The reason of implementing this module for allowing exchange of information is based on the layered communication approach which has been described in chapter 2. Thus the Interface acts as a filter between the external world and the local environment (the Device). However the Production Control module relies heavily on the Database for gathering the information necessary for performing its actions. It was decided to have the two systems running

on the same computer, as communication between the two modules through the Interface would have be inefficient, and therefore a direct connection seemed a more sensible choice. For this reason a direct link between them has been introduced in the Object Model diagram.

Once the top level of the Object Model has been defined it is possible to describe in more detail the different components and the main attributes which characterize them.

The CAD system represents the source of the data which is stored in the database system. The assumption is that if such a system is provided then the appropriate procedures for the transfer of the data from the CAD to the Database and vice-versa is taken. Since the presence of such a module is not relevant for this project it is not developed any further.

The Component class stores the information relevant to components, such as `name`, `quantity`, `location`. This is dynamic information which is necessary to the Production Control Model in order to generate the schedule and coordinate the industrial process.

The DataBase system represented through the DataBase module is the repository of the data which is necessary for the cell to perform its actions. The information about the devices and the components is located here. The object model representing the DataBase determines the way in which the information will be stored in the database internal tables. Since the desire is to develop a framework for a manufacturing system for the footwear industry which ideally could be used for automating any group of operations, the DataBase object model must be general enough to cover all the possible processes. In order to achieve this it had been necessary to understand the manufacturing processes involved and isolate their essential features.

First of all the *DBMS class* can be expanded into the communication subsystem, which permits the network activities, and in the Database Management System. The communication subsystem is identical to the one used for the Device module and it will be analyzed later. The Database Management System is external to our discussion since it represents the database software selected for

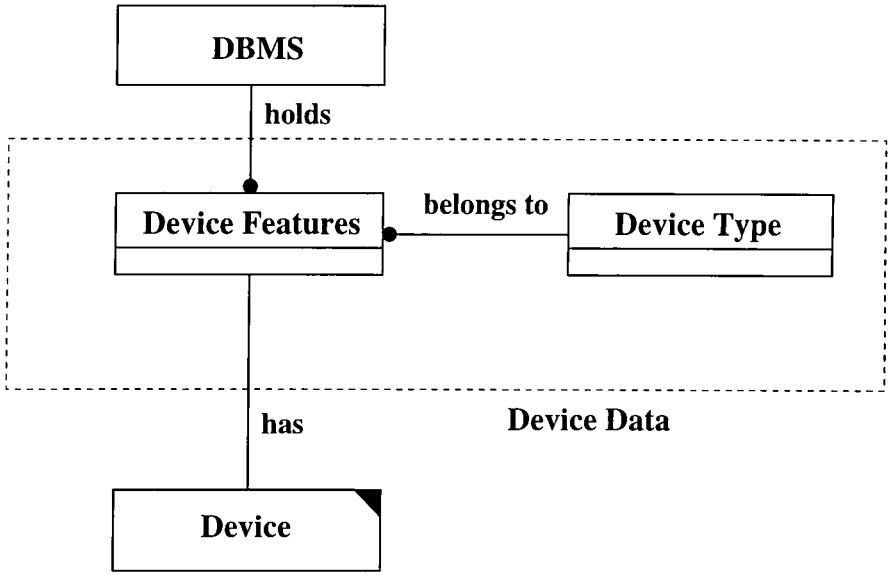


Figure 7.4: Device Data module expanded into its components

the application, and it cannot be further modeled. The Device Data module can be expanded into two components as shown in figure 7.4; the Device Features and the Device Type classes. Each device in the cell has an entry in this class where all the relevant information is stored. Example of attributes of this class are `device_name`, `software_version`, and `last_revision`. The Device Type class identifies the attributes which are common to a set of devices which perform the same task. The Process Time class holds information about the processing times of each component. This information is essential for the generation of the production schedule. The Component Data module groups all the classes which store information about the components processed by the cell. It can be expanded into three elements as shown in figure 7.5: Component Feature, Program Data, and Position. Component Feature and Program Data are abstract classes. The subclasses of Component Feature store the data which is necessary to the device for identifying an incoming component. So far component identification in the footwear industry has been performed using a vision system which uses a moments based approach to differentiate shapes. Using this technique every component belongs to the class of *bad* or *good shape* according to its features. This is likely to change in the future and therefore it is necessary to provide the database with a structure that allows the introduction of new classes which will hold the information necessary

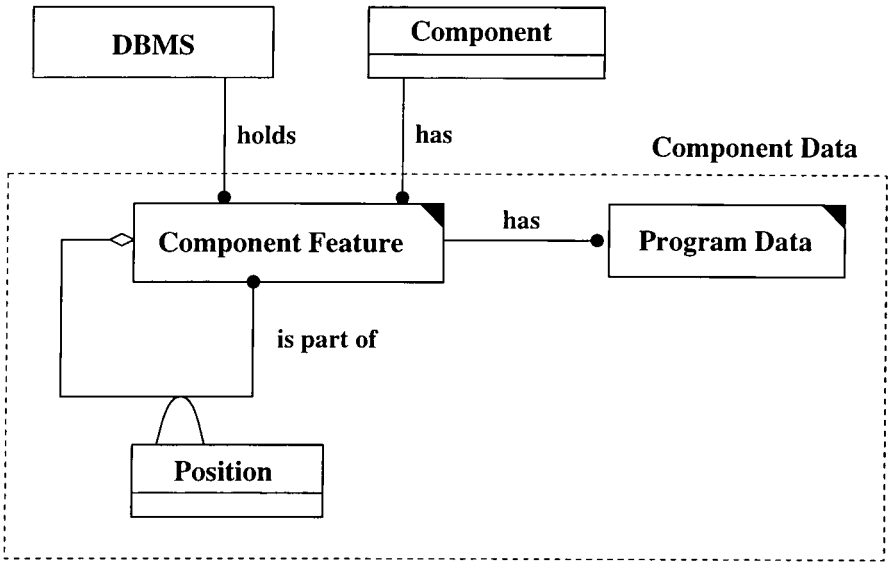


Figure 7.5: Component Data module expanded into its components

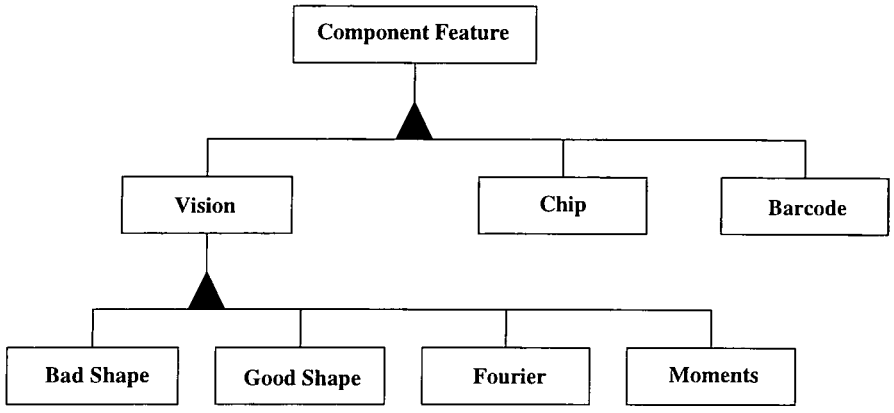


Figure 7.6: Example of expansion of the Component Features abstract class taking into consideration several types of recognition system

for new identification systems. Using an object oriented approach it is possible to define new classes under Component Feature, thus generating a structure which allows the introduction of data structures for other recognition systems. In figure 7.6 a tree structure developed starting from the Component Feature abstract class is displayed. It is possible to see how the database can allow the introduction of the data relevant for devices which use several vision approaches for identifying the components as well as barcodes and chip recognition. Using this approach it is easy is to add to the existing structure new classes for providing the necessary information for recognition systems. A component may have multiple entries under the Component Feature class since the same component may be processed by

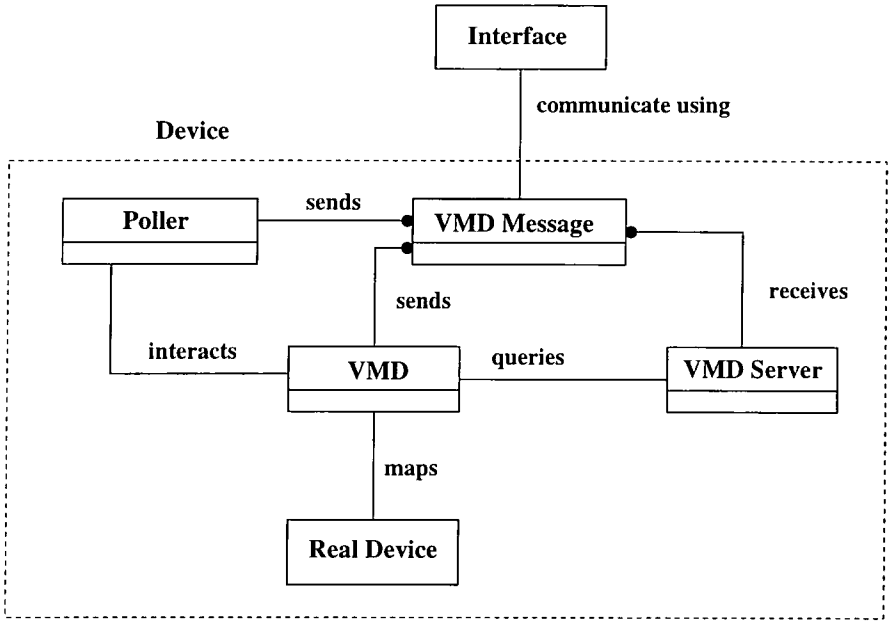


Figure 7.7: Device module expanded into its components

more then one machine which uses different identification techniques. When a device needs to retrieve information related to a component using its name as key, it needs a way to retrieve from the database only the relevant piece of information and to ignore the rest. The `method` attribute which is defined in the Device Type class has been introduced for solving this problem. When a query is done the name of the device identifies to which class it belongs and allows the DataBase to use the associated `method` attribute to restrict the query only to the relevant branches of the Component Feature tree structure. The Program Data abstract class has subclasses which store the program data. For each type of Device Type instance there is a corresponding Program Data subclass which stores the data necessary for the industrial device to perform the appropriate operations on the component. Finally the Position class captures the relationships among components and allows them to be scheduled in the proper order if they need to be joined together during the processing operations.

The Device module represent the devices used in the manufacturing cell. Since the features of the industrial machines employed may be quite different, it is necessary to abstract the common features. This activity is particularly important since it allows the identification of segments of software which will be common to all the devices, thus reducing the amount machine dependent code. The Device

module can broken down as shown in figure 7.7: the VMD (Virtual Manufacturing Device), the Real Device, the VMD Server. the VMD Message and the Poller. The VMD, the Virtual Manufacturing Device, is the most important component of the Device module since it is the one which exchange instructions with the the Real Device and holds its virtual mapping which can be seen by the external world. All the values and status variables which characterize a device are represented as attributes of the VMS class. The class manages all the activities which are necessary for the correct operation of the device in the interconnected environment. It interacts with the external world using messages, which are encoded as instances of the VMD Message class. They are sent to the Interface which forwards them along the network. The incoming messages are received by the VMD Server class which decomposes them in a format which can be processed inside the VMD. The division between the VMD and the VMD Server is useful since the second element is common to all the devices, and therefore can be developed in order to be used by any device as a common piece of software. The VMD on the other hand must be specifically tailored to the device and software reuse is usually not possible. The Poller class performs periodical network activities such as querying remote devices or reporting internal information. Its activities depend greatly on the single device requirements.

The Interface represents a module common to every entity which allows them to exchange information and data. It sets the communication rules which must be followed by every device in order to achieve a successful exchange of messages. The Interface module is expanded in its parts in figure 7.8: the Interface Client, the Interface Server, and Message. The Interface Client processes the VMD Messages from the Device and sends them encoded as instances of the Message class along the network. On the other side the Interface Server receives them and transforms them in VMD Messages which are forwarded to the Device.

The Production Control module groups all the set of systems which coordinate the production activity of workcell (see fig 7.9). It can be subdivided into two main submodules; the one which hosts the control procedures and the communication one. This subdivision allows the system to be modeled using the same approach

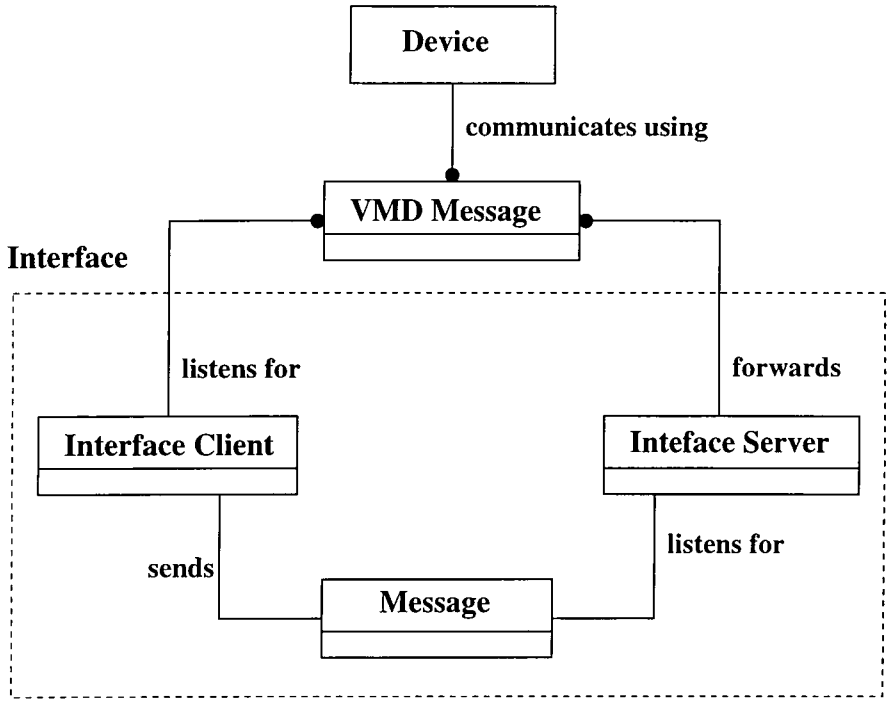


Figure 7.8: Interface module expanded into its components

as used for the Device module. Therefore the communication modules developed for the Device module can be used inside the Production Control one as well. This limits the complexity of the system since it reduces the number of modules which have to be developed and allows software reuse. The workcell control activity is divided into three modules; the Monitor, the Scheduler and the Dispatcher. The Monitor stores an internal representation of the manufacturing workcell. It works in close connection with the Poller which provides periodical updates. The internal structure of this object depends on the controlled system. The Monitor has knowledge of the Component class which stores information about components, number and location. The Scheduler organizes the instance of the Component class in jobs, checks the dependences, and their number, retrieves the timing information from the database and finally generates the schedule. The Dispatcher organized the real-time scheduling activities of the workcell. It invokes the Scheduler when the order to manufacture new components is introduced, and sends the jobs in production relaying on the information about the status of the system held in the Monitor.

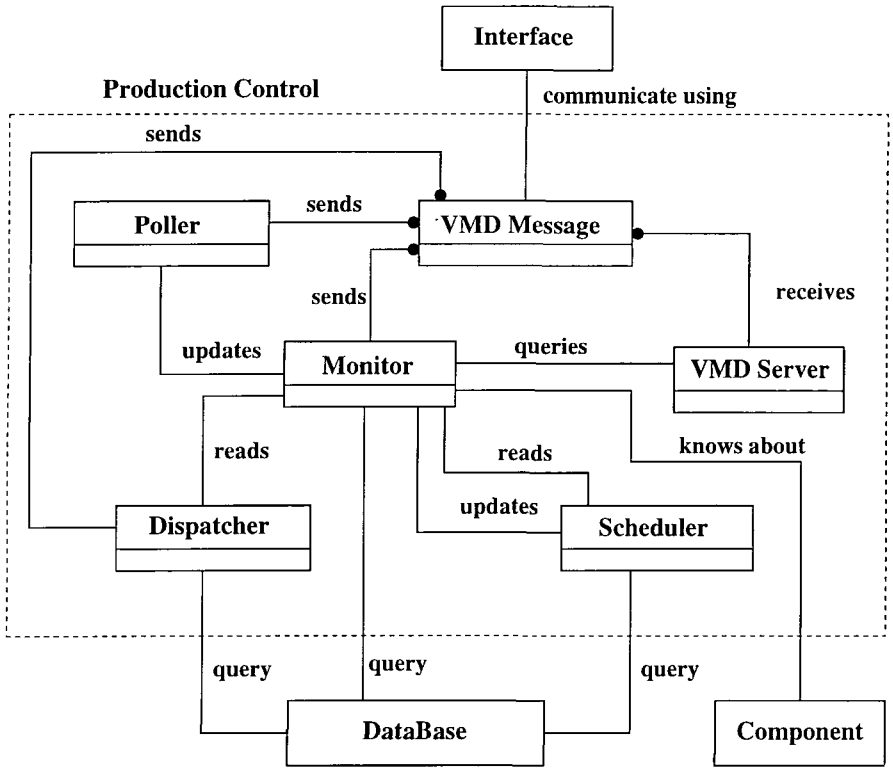


Figure 7.9: Production Control module expanded into its components

7.3.2 Dynamic Model

Once that the Object Model has been defined it is possible to develop the Dynamic Model. As already mentioned it captures the aspects of the system which are subject to modification during the execution of the application. Therefore the Interface, the Device and the Production Control System each have associated one of these diagrams. The model is insignificant for a purely static data repository, and in this case the Database does not have any Dynamic Model (with the exception of the subsystem which handles external queries).

The development of the Dynamic model starts with the identification of the events, externally visible stimuli and responses, which characterize the application. In the case of the Interface the events which characterize the application are the messages. They, in the form of VMD Messages, are sent by the Device and received by the Interface Client which transforms them into Messages and send them along the network. On the other side the Interface Server receives them and transforms them into VMD Messages before forwarding them to the Device. It is therefore

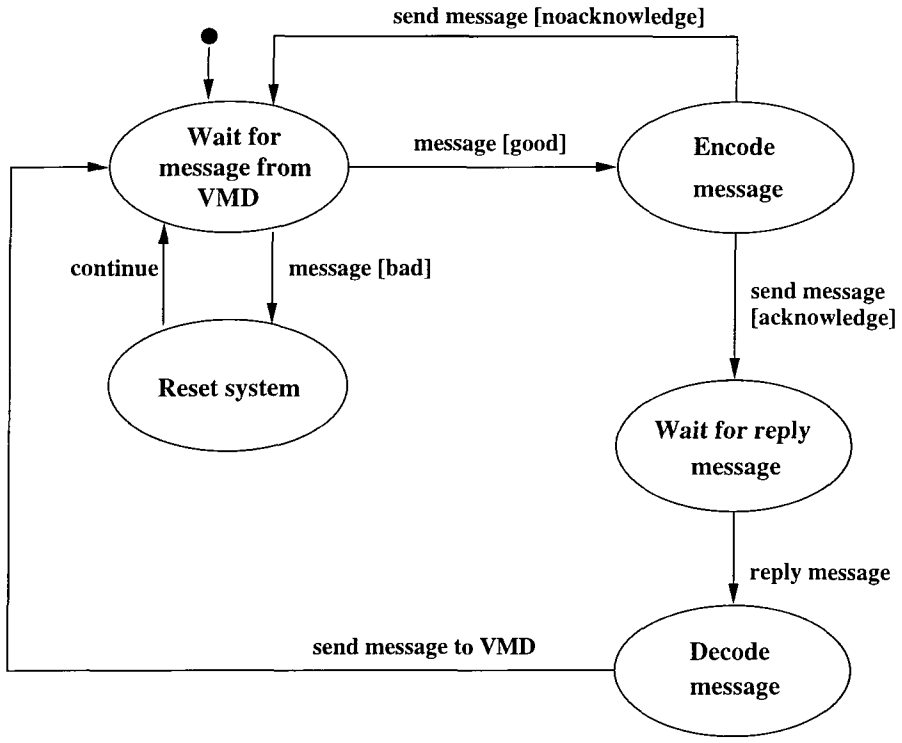


Figure 7.10: Dynamic model of the Interface Client

possible to identify two dynamic entities at the level of the Interface, the Interface Client and the Interface Server. The dynamic model of the first is represented in figure 7.10. After the initialization procedures the Interface Client sits idle waiting for a VMD Message from the Device. When a new message arrives its format is checked. In the case of a bad message, it is discarded, the system is reset, and the Interface Client waits for another message. If the message is recognized as a valid one it is encoded in the proper format before being sent along the network. If the message does not require a reply message or an acknowledge the Interface Client returns to its initial state after sending it. However if a reply message is required it sends a message and waits for the reply. When the reply message arrives it is decoded into VMD format and forwarded to the Device. Then the Interface Client is ready to accept another incoming message from the Device. The behaviour of the Interface Server is similar to the one of the Interface Client and it is represented in figure 7.11. The Server side waits from messages from the Network, decodes them in VMD format and, if valid, forwards them to the Device. If a reply is needed it waits for the VMD Message and, after encoding it, sends it back along the network.

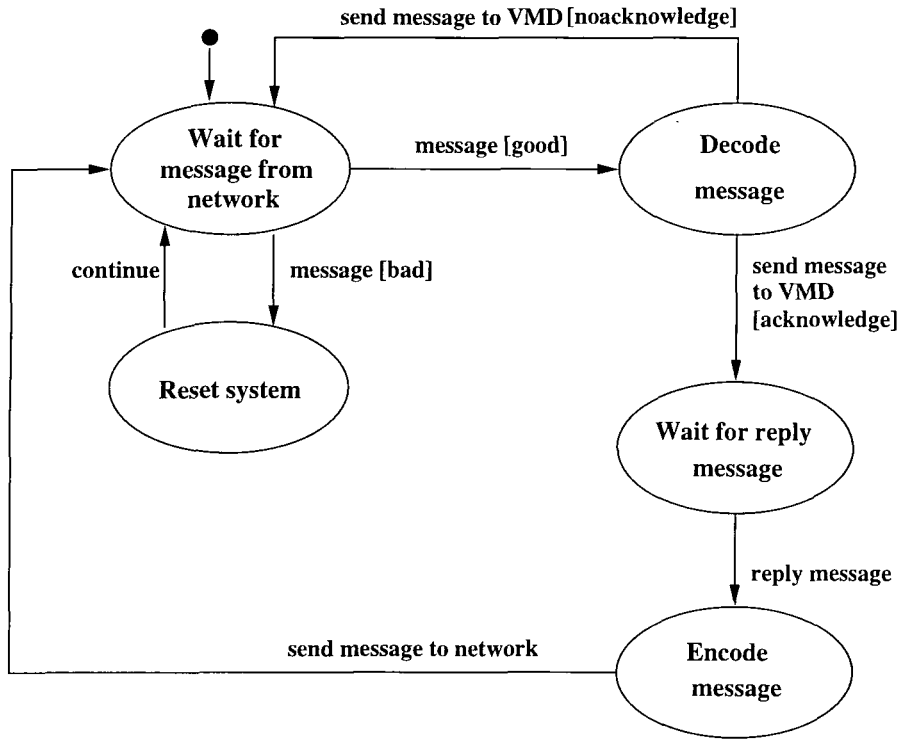


Figure 7.11: Dynamic model of the Interface Server

The development of the dynamic model for the Device is similar to the one of the Interface since the events can be identified as messages. The dynamic model of the VMD Server is very similar to the one for the Interface Server and it is illustrated in figure 7.12. The only noticeable difference is that once that a message has been received and decomposed into its parts it is passed to a function. This interprets it and performs the appropriate set of operations on the object VMD and eventually produces the required answer message. It is not possible to develop the complete dynamic model for the VMD since it depends on the specific application and must be tailored in order to meet the device requirements. However it is possible to model the part of the VMD which is used for sending the VMD messages to the Interface. Its diagram is shown in figure 7.12 and its behaviour is similar to the one for the Interface Client. The events which characterize the Poller are the periodical expiring of the timers which indicate that an action has to be taken and a message sent (see figure 7.13). First the Poller is initialized by reading information on which action has to be performed and at which intervals, and inserts them in a priority queue. Then it enters a loop where the first element of the queue is retrieved. This

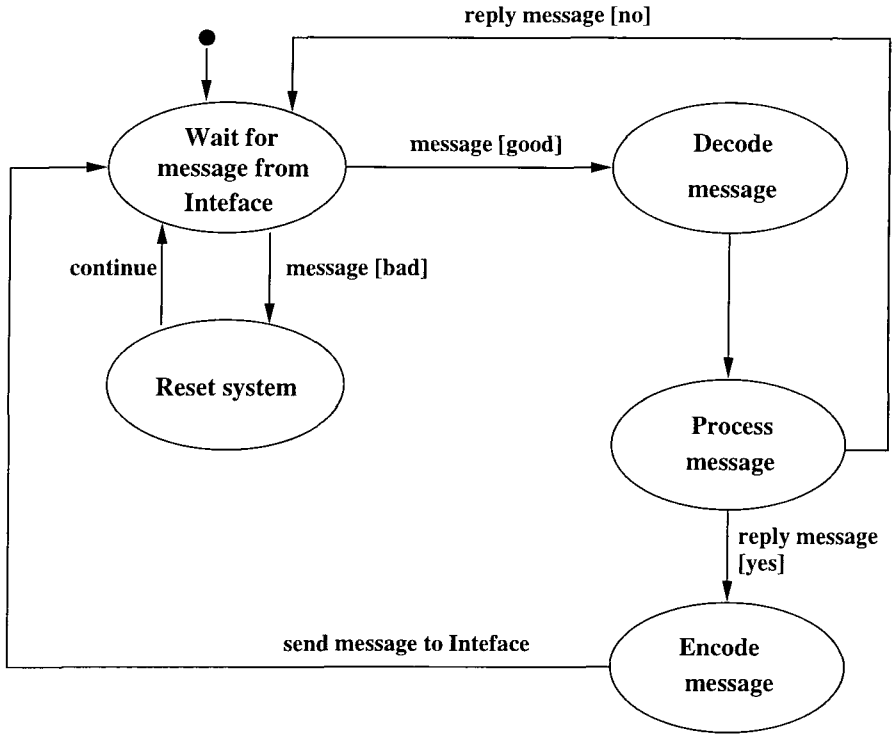


Figure 7.12: Dynamic model of the VMD Server

indicates when the service must be performed and places the Poller in a wait state for the required amount of time. Once this time expires the appropriate action is performed and the queue is updated. Then the cycle starts again.

The Production Control system shares several entities with the Device module and therefore they present an identical dynamic behaviour. Since the Monitor class is a data repository similar to a database for storing dynamically changing data, it does not have a meaningful Dynamical Model. The Scheduler class performs only a transformation on data and it is not affected by external events while it generates the schedule. Therefore Dynamical Model representation does not provide useful information. Only the Dispatcher has a meaningful Dynamical Model representation which is shown in figure 7.14. When it starts, it starts the scheduler and synchronizes with it waiting for the schedule to be ready. Then the first job present in the schedule list is retrieved and the activity of the Dispatcher begins. The starting production times of each component part of the job is calculated. It waits until the first component is due to go into production and wakes up reading the state variables of the system. If all the devices are ready, it informs all the involved devices that a new component is due to arrive and be processed. Then it

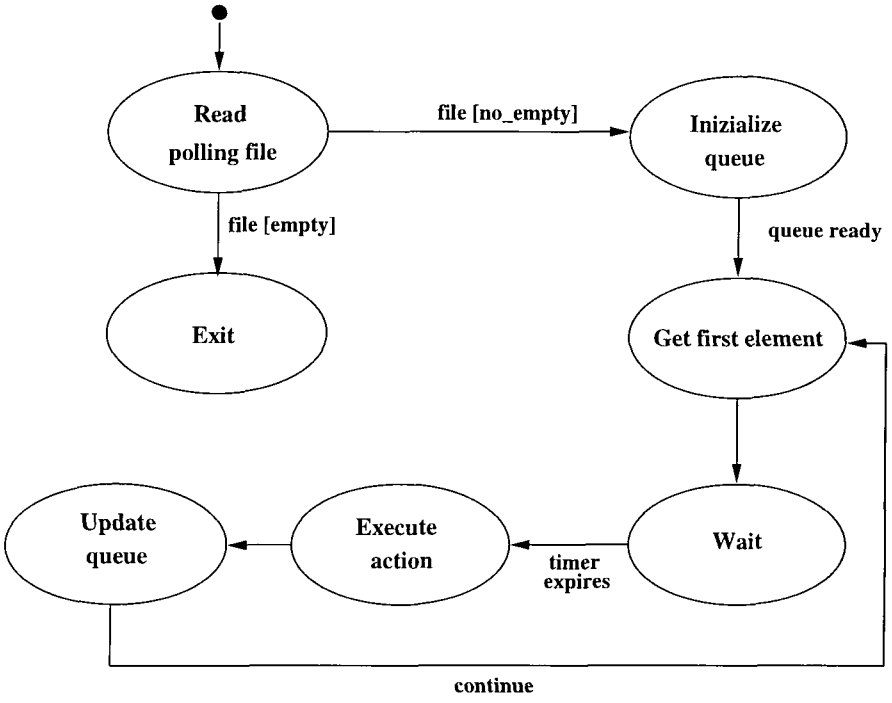


Figure 7.13: Dynamic model of the Poller

performs the same actions for all the components part of the job. Then when a job has been completed it checks if new components to be processed have been added and if so the scheduler is called, otherwise a new job is retrieved from the schedule until exhaustion of the scheduled jobs.

7.3.3 Functional Model

The last stage of the Analysis stage in OMT consists in the definition of the Functional Model. Its aim is to show how values are computed, without taking into account sequencing, decisions, or object structures. In the proposed system it is clear that this model does not have a relevant importance, since it is event driven and no relevant computation is performed. The only exception can be found in the production control system where the algorithm which generates the schedule is performed. The formulation of the scheduling algorithm has already been given in section 6.4. The actions performed by the algorithm are quite simple. The names of the input components are read. Then a query to the database is made in order to find the component relationships and a job for each component to be produced

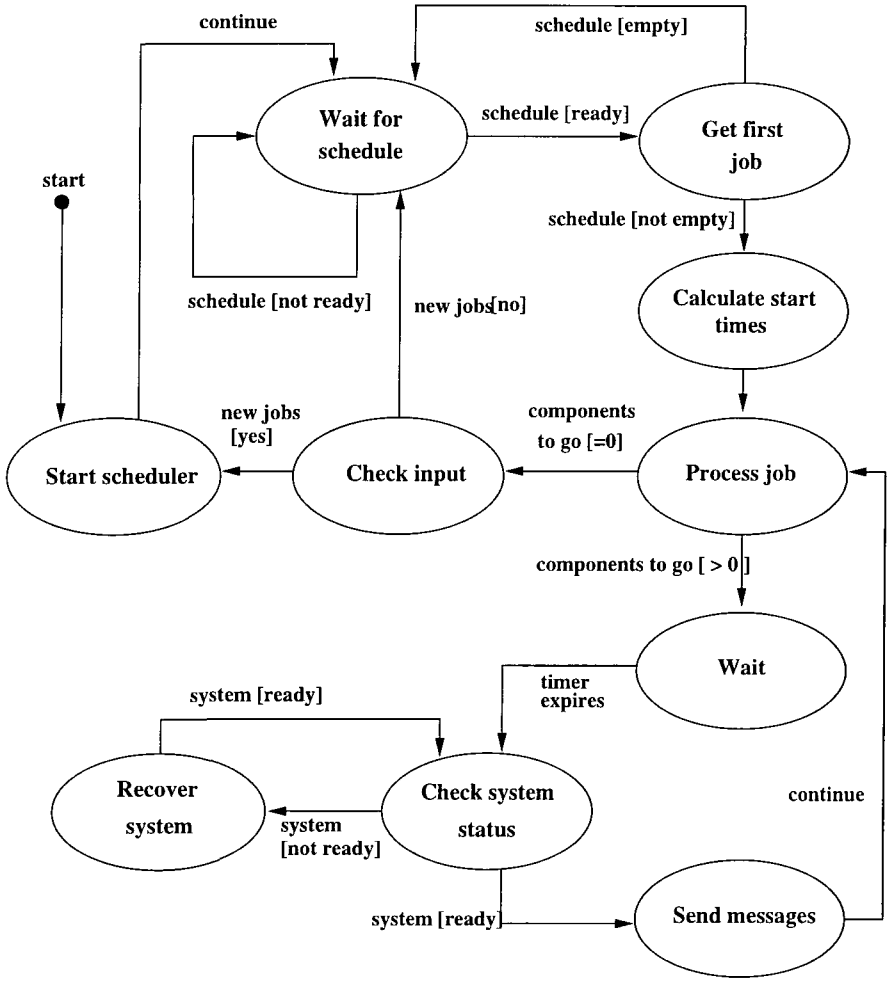


Figure 7.14: Dynamic model of the Dispatcher

is created. Finally the list of all the jobs is sent to the scheduling algorithm which orders them in the appropriate way.

7.4 Conclusion

The design of the system using OMT enables a modular approach to be applied to the problem which greatly simplifies the task. The approach has been shown to be extremely powerful and able to encompass the most important aspects of the system. However it lacks some important support features which are essential during the project design. One of the key aspect in the design approach consists in dividing the system into tractable subsystems. This approach is stressed in the OMT manual, and explicit notations and examples of this are given for the Dy-

namic Model and the Functional Model. However when the Object Model level is formalized, it is stated that it is possible to divide the system into modules, but no notation or practical examples of how to achieve this are provided.

The level of abstraction provided by the Object Level provided the extraction of features common to several components of the system. This is important since it permits the identification of segments of code which can be shared by more than one component, reducing the overall amount of code. The event driven representations which characterize the Dynamic model are particularly useful in the definition of the dynamic relationship among the components.

During the development of the system the use of OMT for the definition of the communication protocol presented difficulties due to the generality of OMT's approach. Since it was not possible to identify any existing methodology for protocol implementation, a framework for their development has been created and is presented and applied in the next chapter.

Chapter 8

Communication Protocol Design

8.1 Introduction

Recent technological developments in digital communications have increased the diversity of the techniques by which communication can be performed. These advances have been matched by the proliferation of communication protocols. There are national and international organizations all over the world whose primary task is to provide sets of standards with which equipment must comply in order to be able to communicate with another device. ISO at international level, IEEE and ANSI in the United States, CENELEC in Europe, DIN in Germany, and BSI in England are some of the well known organizations involved in this standardization effort.

Although there is a large amount of scientific and technical literature related to transmission protocols, industrial specifications and performance evaluation, there is little directed at the theoretical or practical means of creating "first drafts" or initial specifications for protocol design. Much of the published literature covers protocol verification (e.g [Peh90],[SSM90], [Gou93]) and automatic implementation by means of a specialized language or development tool (e.g. [ABM87], [PS91], [BWWH88], [IIK+91], [CLV93]). These methods start from a specification of the protocol needed. Tutorials on protocol specification, such as [Liu89] or [Boc90], address only briefly the issue of how the protocol definition is derived from a con-

sideration of the system requirements.

During the course of the research the problem became evident and it appeared that general design methodologies used to develop software application, such OMT, provide only general support. In order to create a methodology targeted to protocol design and thus covering this gap, a framework, called Rapid Protocol Development(RPD), was developed. Its main aim is to provide guidance in the process of developing new protocols.

The principal reason that a design methodology is needed is due to the fact that in certain industrial applications it may not be possible or convenient to use a standard transmission protocol to implement a communication link between devices. No suitable protocol may exist, or the existing ones may be too complex or inefficient for the application. The purpose of RPD is to provide a framework for assessing the particular requirements of a specific application, and to lead to the generation of fast and efficient protocols for specific connectivity problems. The protocols generated by this approach are particularly suited to small and medium scale implementations where a full-scale commercial protocol might be considered too elaborate or expensive. In this way, structured communication protocols can be implemented even in small systems with low-memory and low-CPU power machines, allowing for economy and simplicity of operation.

8.2 RPD structure

In a typical production environment, machine tools and their controlling equipment will have a limited spectrum of communication requirements, and there may be particular constraints on the capabilities of the equipment available, e.g. limited memory (constraints due to the microcontroller capabilities) or limited bandwidth (noisy lines or very low speed connections). In certain applications, microcontrollers may be used both for the machine control and for communication handling. This could possibly restrict the CPU time available for servicing the network and would demand a simple protocol implementation. One of the reasons that general purpose

object oriented protocols such as MAP [Gen87] have not found wide acceptance in industry is that their level of complexity is not usually necessary in small and medium size industrial plants. The RPD technique seeks to provide guidance in the task of identifying the features needed in an industrial communications system for a specific application. It is suggested that there are three criteria that meet the fundamental needs of good industrial protocols:

- The method should identify the features needed in the protocol
- It should guarantee that there is full expandability
- It should incorporate a recursive approach to problem solving

The technique draws on the scheme put forward by Pimentel [Pim90] by dividing the design process into separated steps:

1. *Identify the communication problem as fully as possible*: This step is used to focus on the problem and to identify the communication requirements of the production plant;
2. *Evaluate the minimal requirements*: This step is introduced to define the minimal needs for the communication network and the possible extensions, evaluated with a minimum cost approach;
3. *Decide the type of the network starting from an analysis of the plant topology*: This step is used to survey existing hardware and software options, and to identify which parts of the protocol have to be implemented;
4. *Identification of command clusters*: This step is used to identify within a functional unit the groups of commands needed to implement the functions required, and to identify common groups of commands between units.
5. *Generate several sets of commands (command categories)*: This step is used to identify the command categories later used to generate the body of the protocol draft;

6. *Decide data fields and assign a code space*: This step defines the data fields which are to be implemented and their format, and uses data from the third and fifth steps to evaluate the size and the characteristics of the code space;

7. *Assign individual codes to messages and functions*: This step maps the code space to the command space in a defined and unambiguous way;

8. *Test the proposed solutions*: In this step the proposed protocols are tested against the system requirements with the assistance of analytical models or software packages;

9. *Select the best solution*: In this step the protocol which best fulfills the system specifications is selected.

Initially, several alternative approaches may be generated, and evaluation of these involves a critical appraisal of the pre-defined criteria. Later, analytical models of the system can be tested by software simulation.

8.3 Protocol development

Since several well defined protocol standards are available in the marketplace, it is important to match the facilities these can provide with the needs of the application. In some cases these protocols may be over-specified, or provide a considerable amount of redundancy for the specific application. In cases where controllers with limited computing power and storage are being used, the RPD technique can identify the critical features that the protocol should possess. It may be that an existing hardware based protocol can be integrated with a specially developed software system. If this can be done, then the new protocol reduces the development time.

There is an attraction in using existing hardware systems standards for the lower levels of the protocols within the OSI/ISO protocol stack, and developing purpose designed application software. This usually requires the adoption of the two lowest layers of the OSI stack, the physical and link layers.

Sometimes for efficiency reasons in the implementation of industrial protocols

only the lowest levels on the ISO/OSI stack are preserved, while the functions required for the specific application protocol, usually provided by the upper layers, are compressed into a new unique link layer or layers.

For example, networks based on a simple RS-232 connection are hardware specified (layer 1), and all additional layers must be developed by the user. If an Ethernet (IEEE 802.3) [IEE85a] or Token Ring (IEEE 802.5) [IEE85b] system is proposed, then the lowest two layers are defined and the user has only to develop the upper part of the protocol.

If we decide to implement a transmission protocol where only the physical layer is defined, then in addition to the whole of the data area of the packet, the fields which would normally have been part of the data link protocol (layer 2) must be defined. A list of the fields commonly found in a communication packet is given below:

- Synchronization heading, [SYNC];
- Addresses (sender and receiver): this requires an additional stage for the evaluation of the required address space [ADRS], and address resolution [ADRS];
- Additional information (sequence number and/or time stamping): for logging, bug-tracking purposes or statistical analysis, [SEQN];
- Control codes, [CONT];
- Error correction codes, [CRC];
- End of packet codes, [EOP];

These basic fields ensure that successful transmission of data between the sender and the receiver can be achieved. The number of fields, and their length are application specific. A packet containing the above mentioned fields may be represented as:

The complexity and sophistication of the packet fields depends to what extent the hardware levels are present in the final system. If only the lowest physical layer is used, then for shared bus operation, the software has to provide a *Medium Access Control mechanism* (MAC), (otherwise provided in the data link layer). A simple polling mechanism might be implemented, and several other options are available.

If the two lowest layers of hardware protocol are used, then the software must implement the *content synchronization mechanism*, which is necessary for the exchange of messages which have been fragmented, and for the acknowledgment of the receipt of packets, and also the *dialog synchronization mechanism* which is used to establish the connection between the sender and the receiver. A complete description of the different techniques used for the implementation of these types of mechanisms is given in [Fre88].

8.4 RPD development guidelines

A set of rules is proposed which enable the design to be carried out in conformance with the steps outlined above.

1. Every packet should conform to the following format of the data field:

   - Packet Identification: a single or multibyte code for packet content identification (this information may be included in the control codes field if the whole packet structure is defined);
   - Message Priority Code: this information can also be included in the control codes field;
   - Data Structure: a data structure with a format of <datalength> + <data>. The length of the message which can be sent in one packet depends on the capacity of input buffer of the receiving device and is specified by the <datalength> field. Data extension should be allowed for in order to transmit messages which cannot fit into one packet. For example, if

the <datalength> field is represented using 2 bytes then up to 65536 bytes of a data field can be specified. A longer data field would require a transmission strategy which breaks the original data into smaller sections. The limiting element here may be the size of the input buffer of the receiver, unless the data acquisition process in the receiving computer is faster than the transmission rate.

2. Code Clustering: codes should be assembled into functional clusters, to provide a hierarchical grouping of similar messages and services.

3. Flexible code space: codes should not fill the entire code space to allow for future enhancements or changes.

4. Packet addressing: multiple addressing should be allowed, for single cast and multicast packets. An example of multiple addressing system is:

   - for 253 nodes: the address is 1 byte plus special codes:

     FF + other address (in order to have other 253 addresses, etc..)

     FE + bitfield (32 bytes) for multicast to the first 253 nodes

     FD + bitfield (32 bytes) for multicast to the other eventual nodes

   - for 65500 nodes: the address is 2 bytes plus special codes:

     FFFF + other address (in order to have other 65500 addresses, etc..)

     FFFE + bitfield (8 Kbytes) for multicast to the first 65500 nodes

     FFFD + list of addresses for multicast to selected nodes

     FFFC + pointer to an table internal to the machines (for addressing topological groups of machines)

It is good procedure not to acknowledge messages which are multicasted in order not to flood the network with these packets.

5. Data representation: when devices which use different internal data representation are present on the same network it is necessary to provide support for transmitting messages in an uniform way.

6. Efficiency criteria: the aim should be to produce a protocol which favours small packets, fast response times, and a small number of addresses, while allowing for future growth and development of the system.

8.5 Communication protocol design

The steps included in the RPD methodology presented in the previous section will be now used for the design of the transmission protocol of the proposed shop floor system.

Step 1

In the first step the communications needs of the system are defined. In this case the aim of the communication system is to establish a link to permit the effective exchange of messages among all the entities connected to network. In detail it is necessary to interconnect a number of industrial machines and robots with a planning system and a database system. The scope of the planning system is to coordinate the overall activity of the manufacturing cell, issuing the appropriate commands when needed. The database has to provide numerical control programs on demand as well as other necessary information to the correct operation of the production plant.

Step 2

Then it is necessary to evaluate the characteristics and the minimal requirements of the system. From an analysis of the features of the application the following set of requirements have been identified

- limited number of devices connected to the network (less than 256);
- random infrequent communication for data program downloading from the database system;

- the dimension of some of the data programs could be of the order of tens of Kbytes (for the stitching machines);
- messages sent from the control system are of small dimensions;
- machines periodically report their status to the control system using small packets;
- no need of semaphores for synchronization;
- alarm messages should be included in the protocol;
- devices on the network may be busy and not able to receive messages;
- prioritizing of the message would be desirable;
- no strict real-time performance required;
- hardware with different architectures may be used.

The existing industrial machines forming the workcell are at the present controlled by Transputers or DSPs, but the intention of including embedded PC boards for controlling the devices' control screens has been expressed by the company.

In addition to the requirements stated above the general guideline of using existing technology in order to reduce the development costs should be always applied.

Step 3

Then it is necessary to decide which type of network to use for implementing the physical connection among the devices. Since the application is quite complex the use of simple communication systems based on RS-232 or RS-485 should be avoided, since the performance would be poor and the development effort excessive. Therefore a better approach consists in considering one of the existing transmission technologies which have been analyzed in chapter 3. Since no hard real-time requirement are needed by the system, token ring and token bus implementation might not be necessary. The survey of the available fieldbusses highlighted their main features. From this information it is possible to note that several of them can be classified as sensor nets and therefore not targeted for shop floor communication and not suitable for our application. In the previous step it was identified

that one of the main activities of the communication system of the proposed workcell is program data transfer whose dimension may be of the order of 50 Kbytes. Unfortunately high bandwidth is not one of the features of the fieldbusses available on the market at the present. Moreover a more detailed analysis has highlighted the fact that file transmission is not very well supported in any of the commercial systems. This factor led to the investigation of the possibility of using Ethernet for the implementation of the physical medium in the communication system. The result of this research and of the tests that were carried out have been already reported in chapter 3. Since from the analysis of the features it became clear that the association of Ethernet/Internet protocols could meet the requirements of the proposed workcell, it was selected for implementing the lower level of the communication system. The decision of BUSM to include embedded PC on their devices is a very strong point in favour of Ethernet.

The decision to use this technology solved the problem of creating the infrastructure for addressing the nodes, since Internet protocols require an IP number, a 32 bit number, to be associated with each device,

In the next steps, only the development of the upper layers of the protocol will be necessary, since all the lower layer functions are already provided. This approach allows a reduction in the development effort and costs, as off the shelf hardware and software can be employed.

Step 4

At this point it is necessary to define which messages need to be proposed at the top of the hierarchy.

The specification of the system gives important information about which are the possible classes of messages needed for the development of the communication protocol. However useful insight may be obtained by reviewing the sets of messages which are defined in other industrial protocols, and adapting them to the needs of the present system. Two of the most complete and sophisticated protocols for industrial applications are the Manufacturing Message Specification (MMS) [BG91] and the top layer of Profibus [Ben93]. The first one defines a complex set of messages for performing activities on the shopfloor. The second one is essentially a

subset of the first which simplifies or eliminates several types of messages. Therefore the work on the definition of the protocol was performed by using these two protocols as guide to understanding the needs of a shopfloor system, analyzing the requirements of the proposed system, and keeping in mind the features of the Ethernet/Internet association.

Three types of messages can be identified at the top of the hierarchy as necessary in order to allow basic communication between the devices included in the network;

- program messages;
- error and event messages;
- general messages.

The first group includes all the messages related to the transmission of data files. The second embraces every message generated when an abnormal or special situation is present, and also includes control commands. Finally the last group consists of all the other messages which do not fall into the first two sets.

In the MMS and Profibus a special set of messages for establishing and releasing connections between nodes is defined. In this protocol one of the Internet protocols, namely TCP/IP already implements such a type of service. This feature leads to a simplified transmission protocol since connection management issues are automatically handled by the underlying layers and do not need to to be implemented and managed within the communication protocol. This feature of TCP/IP solves the problem related to the fact that the connected devices are not always ready to receive an incoming message, since TCP requires that both parts agree that the connection is established before sending a message.

From the definition of the communication requirements and the analysis of the other industrial protocols it is possible to define the command clusters within each set of message exchange.

Program messages have to be able to carry out all the operations related to the transfer of data files from the databases system to the individual devices. They

must provide features for loading and downloading programs. Profibus supports a file transfer protocol which is quite similar to TFTP [Sol92]. After the initialization procedures the program is segmented into small pieces which are sent one at a time over the network to the receiver; each piece needs to be acknowledged before the next one is sent. This is an inefficient way of transferring information, especially if the size of the packet is small and the files are big. Using the features offered by TCP/IP it is possible to implement a more efficient file transmission protocol. The initialization procedures are still necessary, but after that at the top level the file can be passed to the TCP/IP layer which will take care of fragmenting, efficiently transmitting and reassembling it at the other end in a completely transparent way for the application layer. The messages which belong to the program message set have been designed considering the fact that TCP/IP is used. They follow:

- `INITIATE_UPLOAD_FILE`
- `UPLOAD_FILE`
- `INITIATE_DOWNLOAD_FILE`
- `DOWNLOAD_FILE`

The first two commands are related with the uploading of a file. The exact dynamic of the message exchange is presented in figure 8.1. First of all the client sends a request, `INITIATE_UPLOAD_FILE_req`, to the server requiring a file. If the file is present, and found, its features are returned in the `INITIATE_UPLOAD_FILE_ans` message, in case of errors, e.g. file not found, a `INITIATE_UPLOAD_FILE_err` message is sent. Then the message `UPLOAD_FILE_req` is sent by the client, which indicates that it is ready to receive the file. The file is then sent by the server which includes it in an `UPLOAD_FILE_ans`. The other two commands are similar to the first two; in this case the file is downloaded from the client to the server machine. These commands are required when it is necessary to update the version of the program stored in the database after performing some on-line teaching.

Error and event messages handle the transmission of information regarding the

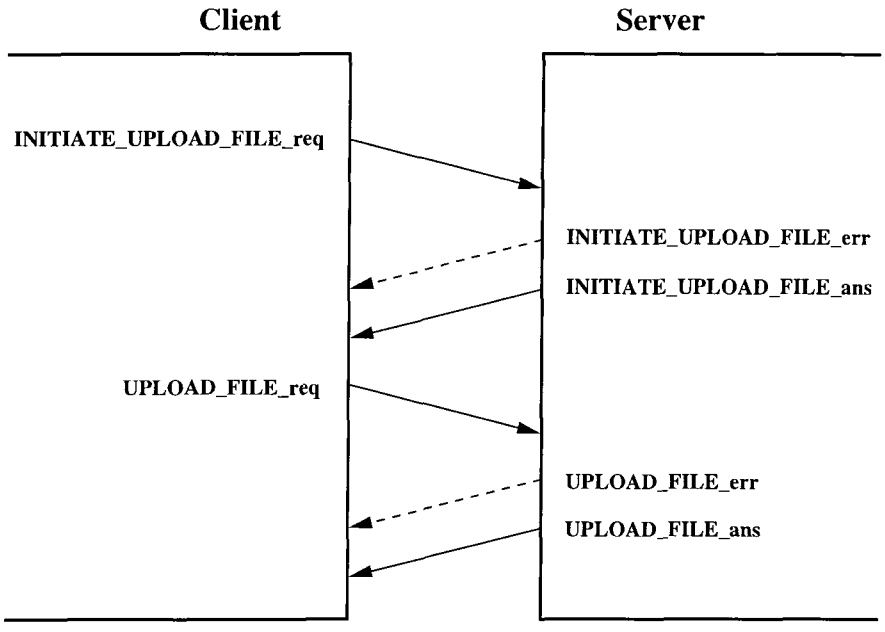


Figure 8.1: Example of exchange of message for performing the uploading of a file from the server to the client stations.

special events inside the manufacturing system. This set of messages also includes messages used to control the execution of the activity of the system. The MMS system used a wide range of messages for event communication and handling, distinguishing between different types of events. This approach was complex and difficult to implement. Profibus reduces the number of the types of events to one, defining only four associated messages. In the present communication protocol only two messages are are defined:

- `EVENT_NOTIFICATION`;
- `ACKNOWLEDGE_EVENT_NOTIFICATION`;

The first message is produced when a special condition arises. Since it may be necessary to broadcast the message, no acknowledge is necessary. On the other hand if an explicit acknowledge is required the message `ACKNOWLEDGE_EVENT_NOTIFICATION` is produced by the receiving device. All the messages related to the the management of the activity of a device, such as starting, stopping, and killing a running activity are included as error and event messages. These are special messages which are likely to be used in initialization procedures, error instances, when something must

be stopped, or during recovery procedures. They are present in both the MMS and the Profibus protocols. The messages defined are:

- START
- STOP
- RESUME
- RESET
- KILL

The names are explanatory of the activity connected with each message.

Finally the general message group includes all the messages which have not been included in the previous categories. They are mainly connected with the activity of reporting the status of the system and reading and writing operations. The messages defined in this set are:

- STATUS
- UNSOLICITED_STATUS
- IDENTIFY
- READ
- WRITE
- INFORMATION_REPORT

The command STATUS is issued when information regarding the status of a device is needed. UNSOLICITED_STATUS is issued when a device wants to communicate its status, often periodically, to another device. IDENTIFY is used to query the identity of a device. READ and WRITE are connected with the reading and writing activities of the system. Finally INFORMATION_REPORT transmits variable information to other devices. The difference between WRITE and INFORMATION_REPORT is that the

latter does not require acknowledgment.

It will be observed that the number of messages defined in the protocol is smaller in comparison to the number defined in MMS or Profibus. This is because the protocol has been structured in a way which is tailored to the requirement of the system, and all the unnecessary features have been excluded. Semaphores have been eliminated since there is no express need for them. However if needed it would be possible to implement them within the `READ` and `WRITE` operation. Moreover the possibility of accessing remote variables using local addresses has not been allowed for, since it defeats the principle of information hiding. Finally the possibility of dynamically creating variables has not been included, since modifications of structures at run time within a critical system could raise security issues. A list of all the message defined in MMS, Profibus and the present system is presented in appendix C.

One of the requirements of the system was the possibility of prioritizing the messages. Giving a different priority to each message would produce complications in the transmission protocol. A simpler approach consists in giving different priorities to groups of messages. In the present case it is possible to associate a different priority to each of the top message groups previously defined. Unfortunately it is not possible to prioritize messages at the Ethernet or Internet protocol level, since messages are processed on *first come first served* basis. However Internet protocol implements a method for delivering messages to specific recipients within a machine (ports). Therefore it is possible to send messages with different priorities to different ports avoiding the problem of jamming important messages in the middle of the incoming queue. The optimal solution in order to achieve this is to implementing the receiver as a multithreaded server application. In this way the system can listen at the same time to more than one port. In addition it is possible to prioritize the different threads of execution and give an higher priority to the one which is listening for the most important messages. In this way if something important is received the less important threads of execution are temporarily suspended in order to service the incoming message. Therefore it is possible to divide messages into three levels of priority. The most important messages are the ones which

belong to the set of error and event messages which have to be serviced as soon as possible since time critical operations may be involved. Then program messages have the second higher priority since the delay in the transfer of a message could make the device incapable of processing the incoming component, thus generating errors. Finally general messages have the lowest priority since they are connected with non-time critical operations.

The proposed shopfloor system is composed of several different devices and it is likely that different architectures might be used within the plant. This, in conjunction with the fact that different programming languages may be used for implementing the receiving and sending interfaces of the devices, raises the problem of differences in data representation. It is necessary to encode the transmitted data in a uniform way which is understood by all the interconnected devices. Since the common format needed could be completely different from the internal formats of the devices it is necessary to divide the communication system into two subsystems: the Device, directly linked to the hardware, and the Interface, which acts as filter, as seen in section 7.3. The advantage of such approach is that the internal and the network data format are separated. The modules included in the Device use the internal data representation. When interactions with the other devices are necessary the Interface acts as a filter encoding and decoding the messages. The role of the Interface is to implement the presentation layer whose task is to provide a barrier to the user application from differences in data representation.

Step 5

At this point it is necessary to specify further the individual services associated with the commands identified in the previous step. It is not necessary to perform this action for all the messages, since some of them might be already fully specified. For example in the protocol developed here when a `STATUS` message is generated a unique answer is produced in response (the physical and logical status of the device are returned). However, in other cases it would be feasible to specify further a set of the commands under the hierarchy of `STATUS` which would request specific information on the status of the system. However messages like `READ` and `WRITE` are likely to be further specified and submessages added. In this protocol submessages under

the WRITE hierarchy have been defined. One of these is used for writing a value into a variable (VARIABLE_DATA), another for informing a device about the features of an incoming component to be processed (COMPONENT_DATA), one for informing the management system of operational times of the device (OPERATING_TIME_DATA) and one for informing a device that a component has been successfully processed and it is approaching (APPROACHING_COMPONENT). Other extensions are possible and the approach is beneficial since it allows the specification of special messages within normal messages reducing the proliferation of special purpose messages at the top level.

At this experimental stage of the development of the communication only the messages which are useful for the modelling of the simulated system have been defined. It is possible that other messages will be defined in order to fulfil additional requirements of the system. It is possible to notice that at this stage of the development of the protocol it is necessary to understand exactly what are the features and the requirements of the underlying system in order to define all the necessary messages.

Step 6

When all the required messages have been defined it is necessary to define the format of the packets and the data fields. Since the transmission protocol is implemented on top of Ethernet and the Internet protocols, only some of the fields need to be specified since the other ones are provided by the underlying layers. Every packet will include the following fields:

- address receiver;
- address sender;
- message type identifier;
- message reference number;
- data length;
- data;

The first two fields specify the addresses of the sender and the receiver; then the message type identifier is included in the packet. A reference number is the next field present; it can be used for monitoring purposes and for verification of reply messages. Then the data length field is included. It specifies the length of the data field which follows. The fields "address receiver" and "address server" are composed of 4 bytes and contain the IP number of the two nodes involved in the transaction. All the other fields are identified with 32 bit integers which should give sufficient space and make the extension codes unnecessary.

An important point is related to the transmission of strings along the network. Because the receiver does not know the dimension of the packet in advance, it is always necessary to transmit the dimensions of the string before sending the string itself.

The first part of the message , the *header*, is common to all the messages and each station knows what to expect when a new message is received. On the other hand the data area, or *body*, of the message is highly dependent from the type of message itself. Usually the message type identifier defines what follows, but further identifiers may be needed to specify what follows if submessages have been defined. In this protocol for the case of the submessages defined under the `WRITE` message hierarchy, a 4 byte integer is expected as the first element in the body of the message. Its purpose is to specify which type of `WRITE` message has been received allowing the correct information to be retrieved.

Step 7

At this stage for each message an identification code is associated. The code must uniquely identify the message. A similar procedure must be performed to define the codes for the submessages. This action is mainly done at the implementation stage and does not present particular difficulties.

Step 8

The communication protocol is then tested to verify its capacity to fulfil the system requirements. This operation is carried out at the during the implementation stage and the testing phase.

Step 9

At the final stage if more than one alternative has been generated the one which is more suited the system requirements is selected.

8.6 Conclusion

The proposed approach, the Rapid Protocol Development (RPD) allows the structured development of a communication protocol, analyzing in turn all the steps from the definition of the problem to the final implementation of the protocol. The technique is designed primarily for implementing protocols in manufacturing industry for custom applications.

The methodology has been applied to the development of a communication protocol specifically tailored for the shoe manufacturing environment, but other examples of applications have been produced [BDP95].

It appears that for many industrial applications, full compliance with ISO/OSI layered protocols is not required, and many cases much simpler protocols are satisfactory and economic for customized systems. However, the method presented here does allow integration with existing protocols if required, and the extent to which this is needed can be identified and incorporated into the model.
Novel protocols can be linked with existing ones giving opportunities for highly efficient application dependent features to be incorporated. The RPD method also provides guidance when an existing system requires upgrading and expansion to improve functionality.

The approach outlined in this chapter attempts to provide a set of guidelines which will assist the designer in producing a communications protocol for a customized application. The immediate application area is in the interconnection of industrial machine cells and their controlling computer systems. Reduction in hardware costs are making this sort of interconnection more attractive, but the software and communication requirements can still appear daunting. The RPD method gives a structured approach to the needs assessment analysis, and leads to the specification of a solution customized to the particular application. However a detailed description of the problems is essential in order to implement a complete

protocol able to fulfill the requirements of the system. The proposed framework allows expansions of the protocol, but as already mentioned, modifications of software after installation may be an expensive procedure which should be avoided.

Chapter 9

Implementation

9.1 Introduction

Once the design stage of the components of the proposed integrated support system for manufacturing workcells has been completed then it is necessary to transfer the classes and relationships developed during the design stage to the chosen programming language and database system. During this process all the decisions related to the system architecture taken during the definition of the communication protocol must be taken into account in order to build a system with the desired features.

Section 9.2 describes the implementation of the database system. Section 9.3 gives an overview of Java, the selected programming language. Sections 9.4, 9.5, and 9.6 present the implementation of the subsystems included in the proposed integrated support system. Then in section 9.7 feedback on the use of Java in developing manufacturing oriented software is given.

9.2 The Database

The database is one of the main components of a Flexible manufacturing system. It stores all the information related to the system and the processed components for prompt access. During the design stage the analysis of the requirements for a

database system for controlling a shoe assembly production plant was carried out. As already stated the main target was to design a system able to meet the requirements of a shoe production line, but at the same time having a structure capable of being adapted to other industrial production systems. In addition the database system required internal flexibility in order to provide ease of incorporation of new features into the system.

Since the development of database system for the storage of the data of the industrial system would have been a time consuming operation, with uncertain results, the decision to use an "off the shelf" database and to interface it with the rest of the system was taken. At present relational data base management systems (DBMS) are gaining popularity at the expense of hierarchical and network DBMS, providing greater flexibility and functionality, but with a reduced performance. The *relational data model* [Cod70], which is built on the simple concept of *table*, is the base of relational DBMSs. A DBMS is essentially a suite of computer programs for managing these tables. The relational database has three major parts;

- data that is presented as tables;
- operators for manipulating tables;
- integrity rules on tables.

Unfortunately advanced applications demand a wide range of capabilities, and no relational DBMS can implement all the features needed to satisfy them. In order to eliminate this problem recently advanced relational DBMSs (also called Object Oriented DBMSs), which implement a support system that enables the introduction of custom features, have been developed. This increase in functionality is gained at a price of loss in performance and security.

Since in the previous chapter the use of an Object Oriented approach had permitted the development of a database with the required features and flexibility, the decision of using an Object Oriented DBMS had been taken. After analyzing some of the available databases which incorporate object oriented features, Postgres95 [YC95], a freely available package developed at the University of California

at Berkeley, has been chosen. It is the result of project started in 1986 [SR85] whose aim was to add substantial features to the existing databases in order to increase their power and flexibility. Now it includes support for classes, inheritance, types, arrays and functions. It is a multiuser database whose query language is SQL92 [Ame92]. Ports for several operating system exists. At present it is actively developed at the University of Berkeley with the support of commercial parties.

Due to the presence of object oriented features in the chosen database, the step for transforming the data encoded in the Object Model in the database code was quite straightforward, since it was possible to preserve the structures identified in the design stage.

The main activity during the implementation of the database consisted in translating the relationships among objects described in the Object Model into SQL code preserving the structure. During this stage *normal forms* [Ken83] were applied for the development of the database. Normal forms are rules developed to avoid logical inconsistencies derived by table update operations. They prohibit different forms of redundancy that could produce meaningless results if one of table is updated independently from other ones. These rules improve database consistency at a cost of increasing the number of tables and slower query execution. The application of the normal forms to the design of the database provided feedback for improving the Object Model.

During the implementation of the database it was necessary to define the fields contained in the tables identified in the object model. Since the exact specifications of the system are unknown only the obvious fields have been inserted. However the addition of new fields in any of the database tables is a simple operation. The identification of the internal and external constraints for each table was necessary in order to increase the internal consistency of the database. This led to the definition of *primary keys*, *foreign keys* and *references* to external tables. Unfortunately Postgres95 does not yet support any type of consistency verification; therefore the relevant lines have been included in the code, but they have been commented.

An extract of the database code showing the implementation of the Component Feature class follows:

```
-----
-- TABLE component: it is the superclass for identifying components.
-- The data necessary to the various recognition system for identifying the
-- components is stored in subclasses of this table.
-----

CREATE TABLE component(
        name                    text,
--        CONSTRAINT component_name_not_null NOT NULL
--        CONSTRAINT component_name_unique UNIQUE (name)
        designer                text,    --the name of the designer
        creation_date           date,    --date of creation
        tot_components          int      --number of components used
                                         --to make this component

--      CONSTRAINT component_pk PRIMARY KEY (name)
);
```

The SQL code defines the class component and specifies four fields. As defined in the design stage this is an abstract class, and subclasses are necessary to completely identify the components. Unfortunately it is not possible to explicitly define abstract classes in Postgres95. Therefore normal classes had to be used, a fact that raises security issues since it would be possible to create entries to the database as instances of this table. In this case the following subclasses have been defined:

```
-----
-- TABLE vision: stores the data about the features used by the vision
-- system of the BUSM for identifying the component processed. It
-- extends the TABLE 'component'. The shape of the component is
-- classified in 'good_shape' or 'bad_shape' according to its features.
-----

CREATE TABLE vision(
                                --Shape features
        root_area               int,    --root area
        mom_max                 int,    --maximum 2nd moment
        mom_min                 int,    --minimum 2nd moment
        g_coeff                 float   --goodness coefficient

)INHERITS (component);

CREATE TABLE good_shape(
                                --Shape features
        radii                   smallint[]     --set of radii
```

```
)INHERITS (vision);

CREATE TABLE bad_shape(
                                --Shape feature
        radii                   smallint[][]    --set of radii

)INHERITS (vision);
```

The table `vision` is a subclass of `component`; it is an abstract class as well. The tables `good_shape` and `bad_shape`, subclasses of the table `vision`, may have entries in the database. They inherit all the fields defined in their superclasses. The name of the component must be not null and unique within a table in order to allow identification by name. The useful feature of Postgres95 which permits storage of data into arrays has been used for storing physical data about the components.

In a similar way all the other classes defined during the design stage were mapped into the database system.

9.3 The programming environment

The main problem related with network distributed applications consists of the fact that the communication protocol used by all the devices connected to the network must be the same. The communication interface which encodes the outgoing messages and decodes the incoming ones must show the same behaviour independently of the underlying hardware and software. This is a demanding constraint and for this reason the first step in order to achieve platform independent behaviour consisted in dividing the communication interface into two layered modules, the virtual manufacturing device (VMD) which maps the underlying hardware and the Interface which acts as a filter between the VMD and the external world. However this leaves the problem that it is necessary to develop new software for every new type of device included in the network.

The solution to this problem was achieved by the decision to use a machine independent language to produce the communication interface and the Virtual Manu-

facturing Device. Java, a programming language developed by Sun Microsystems, represents a new candidate for a robust, architecture independent programming language. Although originally developed for software consumer electronics, it is becoming well known as the language for programming applets, small executable programs which can be embedded into an HTML page on the World Wide Web. It is designed as an interpreted language and this means that the code generated by the compiler is architecture independent, and can run on any system which implements the Java Virtual Machine.

Even if the main use of Java is related to the Internet world, it still keeps the characteristics of a programming language for consumer electronics. The software for these types of applications must have special features; it must be able to be ported on different architectures and processors without changes (as manufacturers tend to change chips quite often as the market introduces new and cheaper components) and it must be extremely reliable since faults may produce failure of the device with possible hazardous consequences.

In "The Java Language: A White Paper" [GM95] Sun describes Java as follows:

> Java: A simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language

Let's analyze in turn all the adjectives which qualify Java [Fla96]:

- *simple*: the developers of Java wanted to keep Java simple easy to learn and use programming language. For these reasons they defined only a limited number of language constructs which are extremely similar to the ones used in C or C++ (and therefore it looks familiar to the majority of the programmers). It has eliminated several of the features of C and C++ which were bug-prone and it has eliminated the use of pointers. In addition Java implements automatic garbage collection releasing the programmer from memory management issues;

- *object oriented*: this means that the programmer is interested in the data of the application and on the methods for manipulating them, instead of thinking in terms of procedures. Almost everything in Java is either a class, a method or an object. Only the most basic primitive operations and data types (int, char, .. ,for, while, etc.) are at a sub-object level;

- *distributed*: this means that it is designed to be suitable for use in distributed applications on networks allowing easy access to remote resources. This feature makes it extremely attractive as a programming language for Internet applications;

- *interpreted*: the Java compiler generates byte-codes instead of executable code (native machine code) and in order to execute Java it is necessary to run an interpreter. Recently however alternatives to interpreted Java have been developed, and Just in Time (JIT) compilers and Java compilers have been introduced.

- *robust*: since it was designed as a consumer electronic programming language, it makes it easier to write reliable software, This is possible thanks to the absence of pointers, due to the fact that Java is a strongly typed language (elimination of type-mismatch problems), and that it performs several run-time checks such as that array and string access are within the bounds;

- *secure*: Java code can be executed in an environment that prohibits it from introducing viruses, deleting or modifying files, or otherwise performing data destroying and computer crashing operations. No one can guarantee that a Java program won't include "malicious" code; Java has created an environment which makes it hard to use the historical techniques which have been used to create dangerous software;

- *architecture neutral*: due to the fact the Java is an interpreted language it can be run on any system implementing a Java Virtual Machine;

- *portable*: Java assures that there are no "implementation dependent" aspects if the language specification (for example the size of an integer is defined at

language level) and only Sun has the right to alter the language. The Java compiler is written in Java while the interpreter is written in ANSI C;

- *high-performance*: since it is an interpreted language it is not as fast as compiled C code, and reports from the newsgroups showed that it is at least 20 times slower than C (some tests on the language performance were carried out as part of this work and are reported in appendix D). However, in future, in the cases where performance is critical, it will be possible to use Java interpreters which perform *on-the-fly compilation* or even Java compilers;

- *multithreaded*; Java is inherently a multi-threaded language. A single Java program can have many different processes executing independently and continuously;

- *dynamic*: it means that Java loads classes as they are needed, and it is possible to link classes dynamically into a running system.

From this description Java has several new and positive features, and it is clear that it is much more than a language for adding animations to Web pages. The next sections will give an overview on the main advantages and current limitations in the use of Java in embedded and real-time systems.

9.3.1 Java for the real world

While in programs like editors and compilers the element of time is not very important, there are large classes of applications which are usually involved with time and deadlines, requiring to some extent real-time performance. There are two main distinct classes of applications, the ones which interact just with the computer and its peripherals, and the ones which need also to interact with the world outside the computer. Representatives of the the first class of programs can be found in the increasing expanding market of multimedia system, teleconferencing and video games. Aircraft control systems, or real-time digital diagnosis systems are examples of applications which interact with the external world. These examples are

often implemented as embedded systems on chips which are merely a component of the whole system.

One of the main advantages of Java consists in the fact that it is a platform independent language. Developers can write an application on one platform and then run it on any platform which implements a Java Virtual Machine. This eliminates the problems related with cross-compiler development systems, multiple platform version maintenance and rewriting and retesting the software every time a port to a new processor is done [Nil96a]. This fact is extremely attractive since programs like video games could be developed on one platform and then be ready to be commercialized for a whole set of computers. In the fast moving scene of embedded systems where new and improved chips are introduced at a high rate, applications which could be simply ported from one chip to another offer immediate advantages in embedded systems. Old silicon could be quickly replaced without the need to spend a lot of time adapting the software to the new chip. The platform independence of Java presents some complications since developers may not be able to test their applications in each environment where they are expected to run. A real-time application would have to run within the same constraints on platform with different features and speed (for example a 33 MHz Intel 40486 and a 300 MHz Digital Alpha) [Nil96a]. Moreover it is possible to run the application as a process on a time shared operating system where it would compete with the other tasks for the CPU time.

Expecting to use Java as the solution for implementing real-time programs which are able to meet the same deadlines on any platform is unreasonable. This is especially true when hard real-time performances are required. It is not possible to expect that a real-time application which just manages to achieve its goals on a fast machine like a Sun Ultra would be able to provide the same behaviour on a Intel 486 platform. In many applications the development of a real-time system must be carried out closely allied to the underlying hardware since performance must be verified and guaranteed to meet constraints. Having a language that can run on any platform does not mean that it will be able to produce the same time behaviour on all the platforms. Testing, and may be modifications, are always a

necessary step when a new porting is performed.

Before starting the discussion it is necessary to realise that the term Java comprises three different entities;

- Java the language;
- Java the Byte code;
- Java the Virtual Machine (JavaVM).

While the first two are strictly defined by Sun Microsystems and any alteration in them would produce the result of creating a dialect of Java, the JavaVM is not completely specified, a fact that leaves some freedom in implementation.

9.3.2 Real-time features

The platform independence of Java is a great advantage of the language, but there are other positive features which are desirable in a language which has to be used for real-time development.

First Java allows easier code development. It is object oriented, but without the complexity of C++. The structure of the language eliminates complete classes of bugs which are common during the development of C and C++ programs (no pointers, no operator overloading or multiple inheritance). In addition its platform independence permits the initial code testing and debugging on other host platforms before starting to work on the target.

Another important advantage consists in memory management, since Java relies on automatic garbage collection while C and C++ generally require the programmer to free up unneeded memory. Memory leaks (and, worse, freeing an object which is still being referenced somewhere) are some of the most common programming errors.

Java's type safety is another feature which should decrease the number of bugs. However at the same time it prevents direct access to hardware registers, as is often

done in C and C++ via casts. Allowing Java to access the underlying hardware would produce two types of problems. First of all it would raise security issues, secondly, since Java is supposed to be platform independent, it would prejudice platform independent features. This could be seen as a problem in embedded applications where it is necessary to access the hardware. However Sun Microsystems has announced the development of a Java version for embedded systems, but full details have not yet been released. At present the only solution to this problem is to take advantage of the feature of Java which permits the inclusion of methods written in native code into Java programs. This means that it is possible to write procedures which access the hardware in some local native system language, typically C or C++, and include them in the Java application. These methods cannot provide the security guarantees typical of a Java program. Moreover native code cannot guarantee portability and any Java code that relies on native methods must be ported to each type of platform on which that code is supposed to run.

Other useful Java features include its fixed-size longs, ints, shorts and bytes which allow the programmer to write more portable code, although the lack of unsigned types may be a problem in some cases.

At a higher level Java allows the change of the code "on the fly" since Java classes can be dynamically loaded into the application. This feature is especially useful for systems which need a lot of flexibility. At present this capability is used for loading applets across the Internet. This ability to add new applets or update old ones on the fly is extremely powerful, and can add to the functionality of any system easily and quickly. Users can get new functionality or new user interfaces by simply connecting to the local server or the network. This is extremely useful in applications like multimedia. However using the same facility in safety critical system could raise security issues. In addition most embedded systems have the problem that once they are deployed in the field, updates to the software often don't make sense without concurrent changes to the hardware. Besides, many are not even connected to any sort of network that could be used as a Java server, but this situation may change in the future.

Java's built in support for multithreading and multitasking represents an ad-

vantage during the development of real-time applications. However the capability of predicting the behaviour of an application written in Java in a multitasking environment requires support from the run time system. One possibility to achieve predictable behaviour consists in developing a JavaVM which takes advantage of the features of a real-time operating system (RTOS). In this context Java with its VM is more analogous to C or C++ running on a RTOS kernel. These other languages provide support for multitasking but it is not integrated in the way it is in Java. At present the developers of the RTOS QNX have expressed the intention to develop a JavaVM which will run on their operating system. However commercial RTOS's generally provide a wider range of features for process control than Java and its class libraries, and therefore some additional code would have to be written to make Java competitive in this market. Perhaps if web applets were not the largest part of the Java applications, this is a direction that the language might have taken. At present the standard POSIX.1b [IEE95b, Gal95] defines a set of support function for the development of real-time applications in real-time multitasking environments. If Java is to be used for developing real-time systems it has to provide a set of functions comparable to the ones defined in the standard. The files `<unistd.h>` and `<limits.h>` describe the operating system's POSIX support at compile time. Since Java is supposed to be machine independent the local configuration of the system should not be an issue of the application, but eventually only for the JavaVM. A function for checking the run time environment, like `sysconf` defined in POSIX, does not make sense in the Java language for the same reason. The problem of *namespace pollution* addressed in POSIX is eliminated in Java since there are no global variables or functions, and a unique package naming scheme that is based on the domain name of the organization where the the package has been developed has been proposed [Fla96]. Java allows the creation of child processes by calling the method `Process.exec()`. The object returned by the `exec()` call provides methods to access the input, output and error streams of the child process. In addition methods for waiting for the process termination, killing it or retrieving its exit value are provided.

POSIX defines a series of mechanisms which allow the communication and co-

ordination of processes; signals, messages, shared memory, and synchronization. Signals are used for many purposes including exception handling, process notification of asynchronous event occurrence, process termination in abnormal circumstances, emulation of multitasking and interprocess communication. Java does not have any support for signals, but several of the tasks implemented by signals can be done in other ways in Java, for example by using the Java built in exception handling mechanism or threads. Signals for interprocess communication cannot be implemented easily (in any case signals are not an efficient and reliable interprocess communication system even if POSIX.1b adds to them some useful features). Messages are used to pass information between processes in a more efficient way than signals. Three different mechanisms for message passing are available; pipes, FIFO and message queues. Pipes work like streams; one process simply writes data at one end of the pipe and the other one reads at the other end. A FIFO is simply a pipe with a name. A message queue is a more sophisticated version of a FIFO since it consists of a priority queue of discrete messages. In Java since it is possible to access the input and output streams of the child process, the implementation of a pipe is a simple operation. If it is necessary to prioritize the messages it is possible to create a thread in the child process which sorts the messages and forwards them in order to the main application.

The creation of named entities is a more problematic issue. In other situations, shared memory provides a low level for processes to communicate with each other. Unfortunately since Java is not able to address memory directly it is not possible to implement shared memory. A higher level solution for implementing shared memory would consist in declaring common segments of memory within a process. However this solution would rely on pipes for communication, and one of the main features of shared memory is to provide fast access to the data. Since it is impossible to implement shared memory in Java at present the problem of synchronizing multiple processes will not be apparent. If a process needs to read data held in another process the second one can decide when to provide the information, since it holds the data required, and the problem of generation of partially updated data does not exist at process level. POSIX provides a set of functions for setting the

scheduling policies of the processes. Since Java the language has no control over the OS where the application is running, it is not possible to define a similar set of facilities. A partial solution to this problem could be implemented at the level of the JavaVM. It could be written in such a way that it would be possible for it to define the scheduling policy and the priority of the process which is going to run when it is started. In this way no alteration of the language would be necessary, but the solution would not provide all the flexibility that the POSIX approach allows.

Java has access to the system clock (with millisecond resolution). It provides basic timer facilities which could be used inside the `Thread` class. These are not as sophisticated as the timers defined in POSIX, but they could provide sufficient support in many applications. Scheduling and timers are the means for making applications perform operations on time. However the OS may create some obstacles to letting this happen. The first problem is related to the performance of the OS. If the OS is too slow the only solution to this problem is to use a faster OS and/or platform. The second problem consists in the fact that many modern operating systems make extensive use of virtual memory. If not enough physical memory is present the system performs automatic paging moving parts of physical memory onto hard disk making new space available. In addition swapping may occur which involves dumping the entire process out to disk and reusing its physical memory. The operating system decides what and when paging or swapping occurs. The problem is that the time of accessing data which has been transferred to hard disk takes an order of magnitude greater in comparison with that when data is present in the physical memory, and therefore the expected response time cannot be guaranteed. The solution is to lock down data or the entire processes in memory. At the language level Java does not allow this. However again the solution for locking entire processes down in memory could be found at the level of the JavaVM which could give the OS the instruction to avoid the swapping out of memory of executed process. Data locking could be achieved by extending the meaning of the `volatile` keyword. At present a variable is defined as `volatile` if it changes asynchronously and therefore the compiler must read the variable's value from memory every time

and not attempt to save its value in a register. The JavaVM could extend this definition by locking down in memory data which is defined as `volatile`.

POSIX defines support for synchronized I/O. When I/O operations are synchronized with the underlying device it means that they return only when the device is appropriately updated. This is not always the case since several systems for efficiency reasons use a buffer cache to hold I/O for later flushing on disk. This provides an increase in performance. These functions are not present in Java, but could be implemented at the level of the JavaVM. Unfortunately in this case all the I/O operations (or classes of them) would be treated as synchronized with associated performance problems. Since I/O synchronized methods are necessary to access devices, the inclusion of a method written in native code using the POSIX features would be a better solution. Finally POSIX provides support of asynchronous I/O, which means that I/O operations are executed in parallel with the application. Java allows this to be achieved using threads.

At the present Java lacks of some of the features required by the POSIX standard, but some of the problems could be solved modifying the JavaVM. However Ken Arnold and James Gosling, the creators of Java, write [AG96]:

> Each implementation of Java must provide one or more appropriate extended classes of `Process` that are able to interact with processes on the underlying system. Such classes might have extended functionality that would be useful for programming on the underlying system. The local documentation should contain information on this extended functionality.

Therefore it is possible to expect some developments in the future in this part of the language, which may however undermine its platform independence.

There are other problems connected with Java which are shared by any type of application designed to achieve real-time goals. Overall performance is an important issue. Real-time is not concerned with execution speed of programs as such, but faster execution allows deadlines to be met more easily. The interpretation of the Java Bytecode makes sense on Internet applications which must run on any

platform, but it is generally undesirable in other cases. However Java need not necessarily to be an interpreted language, even if at present implementations are based on interpreters. There are other two possible alternatives which increase performance. The first option involves the precompilation of Java to native machine code. After a product has been developed, the Java code can be compiled directly to native machine code prior to shipping. At present there is only one example of Java compiler, Toba [PGTNW97], developed at the University of Arizona, but it is likely that their number will increase soon. The second option consists in applying Just In Time (JIT) compilation. This type of compilation, known also as "on the fly" compilation, works in the following way. Once a Bytecode has been loaded into a particular virtual machine environment it can be translated at run time (by the virtual machine environment) into the host machine's target instruction set just before the code is actually executed. An example of such type of JavaVM has been developed with the Kaffe VM [Pro97]. The use of a JavaVM is problematic for embedded applications since they are often implemented on small cheap and power frugal chips. Memory on these systems is usually an expensive option which is always kept to the minimum. Since the code of the JavaVM can be measured in hundreds of kilobytes its use is not a feasible option in small applications. The size of the JavaVM itself is of the order of 64 Kbytes, but standard class libraries occupy a large amount of memory. It is possible to think of implementing a JavaVM for embedded use which does not include the libraries since they are not part of the language, thus reducing the size of the interpreter. The implementation of a JIT compiler is a feasible approach, but only for larger systems since it requires more memory to allocate the Bytecode, the JIT compiler and the translated native code, in comparison with an interpreter or the native code approach. For applications targeted to be executed on small dedicated systems it is possible to say that the best solution would consist of compiling native code since the JavaVM does not provide additional useful features. On the other hand for applications which run on more powerful machines the use of JIT compiler could be the best option since it does not require the developers to compile the code individually for the specific platforms. At the time Java was launched, Sun Microsystems announced the in-

troduction of a series of Java chips, microprocessors with different features, which could run Java Bytecode without the need of a JavaVM. Unfortunately Sun has not released the full details on these chips and therefore their assessment is difficult. However the idea did not find enthusiasts in the area of embedded system since it is not clear how it would be possible to develop device drivers (which need to address the hardware) using the current version of Java. Moreover at the high end chip market it is difficult to see the advantages of a Java chip since Java applications could be happily running using a JIT or recompiler. Even if Java chips provide a good performance, they will still have to compete with the economies of scale of the general-purpose microprocessor industry and in the past similar approaches have proved not to be successful (e.g. the SOAR chips).

At present Newmonics Inc. is developing an ambitious project for a clean-room implementation of Java especially designed for reliable embedded real-time systems [Nil96b]. The proposed JavaVM implementation, called PERC, would allow loading and analyzing of the Java code at run time. When a new real time activity is loaded a configuration manager would assess the requirements of the code (execution time, memory requirements etc.). Then a resource negotiator would analyze the data from the configuration manager and decide if the new application can be accommodated. If this is not possible, other applications would be terminated to free resources for the new one. If it is not possible to terminate any of the active applications the problem would be reported. The activities related to the timing analysis of the code and its schedulability performed by the PERC are complex and many people have raised doubts on the feasibility of the project. It seems that the company aims as a first step to implement a JavaVM for soft real-time applications and then to develop a hard real-time version. However the majority of the critics of the project question the decision of the developers to introduce two new keywords; the `atomic` statement for providing a new type of synchronization mechanism and the `timed` one for defining the upper bound of CPU time allowed to the code included in this statement. The reasons of their introduction are to provide greater efficiency in comparison with `synchronized` statements, to make the code more readable, and to simplify the development of the Bytecode verifier.

It is believed that these extensions to the language are dangerous since they will create a non-portable Java dialect. They also seem unnecessary since it would be possible to develop class libraries with equivalent semantics, which used in conjunction with a run time environment able to interpret their semantics, could provide the same functionality. The company recognizes this since it plans to develop a compiler which would translate extended code in plain Java (code or Bytecode), however it is not known if it would be possible to achieve the same performance.

Another example of development of Java classes for the support of real-time applications has been done by James Young at the University of Berkeley. He has developed support for "real-time tasks", which are directly analogous to threads, except that they have attached to them some additional timing information related to worst case execution, expected execution time, and deadlines. It is possible to specify hard real-time, soft real-time and non real-time tasks. Tasks are coordinated by the JavaRealTime executive which chooses to accept new tasks or not. Unfortunately sometimes browsers and JavaVMs running applications which make use of JavaRealTime terminate themselves. The package has been proved to be very reliable on some platforms (such as Pentium 133 with Win95, using Sun JDK 1.0.2), but immediately kills the JavaVM on other platforms (for example DEC Alpha, using Netscape 2.02). In other cases the behaviour is inconsistent and depends on the runs. The author presumes that this may be due to the combination of bugs in the code of the JavaRealTime and bugs in Java VM implementations on the vendors' part.

Summarizing, Java offers several interesting features for the development of real-time applications. At present due to the fact that it is not possible to address the underlying hardware it is more suitable for applications which do nor require interaction with external devices (multimedia, games). The language does not have all the features which are defined in the POSIX1.b standard, but several of them can be implemented at language level, and other ones at the JavaVM one. The performance of the interpreted version of Java is currently slow, but it is rapidly improving with the introduction of native and JIT compilers. There are some examples of projects whose aim is to provide the means to write real-time applications

with Java, but they are still at the development stage. In any case the introduction of new keywords in the language for the support of real-time features should be done by Sun in order to eliminate the problems of possible Java dialects. Extensions and implementation of the JavaVM designed for the real-time time world would be a more satisfactory alternative in order to allow Java to achieve real time performances.

9.3.3 The reasons for using Java in industrial automation

Apart the possibility of providing a platform independent language there are other features which made Java an attractive programming language for the implementation of the proposed support system.

First of all Java is an object oriented programming language and therefore the OMT design can be immediately mapped into a program written in Java. Its resemblance with C and C++ with the elimination of several of the error-prone features of these languages makes it easy to learn and use. For example the `goto` statement has been replaced by other constructs, `structs` and `unions` have been removed. Operator overloading and multiple inheritance present in C++ have been discarded. However the most important simplifications concern the elimination of pointers and direct memory management (allocation and deallocation of Objects) which is performed by an automatic garbage collector. These are features which considerably speed up the debugging of the code, eliminating entire classes of bugs.

Java's built in support for threading and networking was crucial in the selection of this language for the development of the system. The `java.net.package` offers a set of classes for creating and managing network connections, in a simple and direct way. Full support is given for TCP and UDP sockets as well as for managing URL connections. The process for establishing a UDP connection is simple and relief programmers from socket level details. A simple program which sends a specified text as a datagram to port 4567 of the specified host written in C follows;

```
#include<netdb.h>
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>

#define UDP_PORT 4567

main(argc, argv)
int argc;
char *argv[];
{

  struct sockaddr_in client, server; /*server address assembled here*/
  struct hostent *host_info;
  int sock;
  char *server_name, *message;

  server_name = argv[0];
  message = argv[1];

  sock = socket(AF_INET, SOCK_DGRAM, 0);          /* create socket */
  if (sock < 0)
  {
    perror("opening stream socket"); return 0;
  }
  client.sin_family = AF_INET;
  client.sin_addr.s_addr = htonl(INADDR_ANY);
  client.sin_port = 0;                    /* 0 means choose any port*/

  if (bind (sock, &client, sizeof client) < 0)
  {
    fprintf(stderr,"bind failed\n"); exit (1);
  }
  host_info = gethostbyname(server_name);
  if (host_info == 0)
  {
    fprintf(stderr, "%s : unknown host\r\n", server_name); exit(1);
  }
  /*set up the server's socket address, then connect */
  server.sin_family = AF_INET;
  memcpy((char *)&server.sin_addr, (cahr *) host_info->h_addr,
                                          host_info->length);
  server.sin_port = htons(UDP_PORT);

  sendto(sock, message, sizeof(message), 0, &server, sizeof(server));

}
```

It is possible to notice that in order to be able to use a UDP socket in a C program several (error prone) instructions have to be included in the code. A program which performs the same operation written in Java follows (from [Fla96]):

```
import java.io.*;
import java.net.*;

public class UDPSend
{
  static final int port = 4567;

  public static void main(String args[]) throws Exception
  {

      // Get the internet address of the specified host
    InetAddress address = InetAddress.getByName(args[0]);

      // Convert the message to an array of bytes
    int msglen = args[1].length();
    byte[] message = new byte[msglen];
    args[1].getBytes(0, msglen, message, 0);

      // Initialize the packet with data and address
    DatagramPacket packet = new DatagramPacket(message, msglen,
                            address, port);

     // Create a socket, and send the packet through it.
    DatagramSocket socket = new DatagramSocket();
    socket.send(packet);
  }
}
```

From this small example it is possible to appreciate the compact nature of Java code in comparison with C. It is also possible to note how Java wrappers increase program simplicity and readability avoiding programmers having to deal with low level socket details.

The package `java.lang.Thread` offers a set of functions for supporting multiple threads for execution within one program. In this way it is possible to handle more than one task a time. Since threads have been a design point of the language, they are an integral part of the language itself and not an external set of libraries as is true for C or C++. For this reason it is easier to program in a threaded

fashion using Java in comparison to other programming languages. On the other hand Java threads lack of several features and synchronization methods which are provided by POSIX conformant threads [IEE95a]. This reduces the complexity of code and makes threads easier to use and manage, but at the same time they may not provide the level of detail that some applications require. For example in Java the main synchronization mechanism is based on the concept of *monitors* [Hoa74], which serialize and coordinate the thread activities. No support is provided for the low-level, often more flexible and efficient, set of synchronization primitives such as mutex, semaphores, variables, reader/writers locks and barriers. However the built in thread capabilities offer the possibility of creating concurrent and distributed client/server applications in a simple and concise way.

Another strong point of Java consists in the possibility of accessing several useful collections of frequently used structures which are included in the `java.util` package. Some of the most interesting ones are `Vector`, `Stack`, `Hastable` and `BitSet`. The inclusion of this set of generic utilities as a standard part of the development environment simplifies the programming efforts. These useful functions do not have to be developed every time from scratch, as often happens in C and C++ where it is not possible to relay on external libraries, which may be absent on some platforms. These standard utilities are particularly useful for the reducing the amount of code that applets carry along the web, since all standard Java components can be pre-configured into browsers [JS97].

One of the critical issues of the acceptance of Java is the possibility of preserving the vast quantity of software which is not written in this language. In addition to the situation where all the code of the application cannot be written completely in Java, a case where another language may be more efficient may arise [AG96]. In order to cope with these situations Java supports *native* methods which are sections of code written in another programming language, typically C or C++.

However, at present Java presents some limitations which became evident during the assessment of the suitability of the language for the proposed manufacturing support system and are worth restating here. It became evident that the only support for providing synchronization for threads consists in using monitors. In

addition it is not possible to send signals from one thread to another, and there are no facilities for interrupting threads. Timeouts for sockets are not implemented, and it is not possible to send broadcast messages. Some of common constructs used in C and in particular, templates and enumerated types are absent from the language. However one of the most serious problems is the lack of the pointers and the consequent impossibility of addressing the underlying hardware. As already mentioned the only solution to this problem consists in implementing native methods every time a call to the underlying hardware is necessary. This immediately causes the loss of machine independence provided by Java. Moreover since the language is new, there is a lack of support tools for the development of the code such as debuggers. Finally due to the lack of people experienced in the language, support from other developers with knowledge in the field was completely absent.

Analyzing the benefit and limitations of Java the decision to use the language for development of the industrial system was taken. The reason of this decision lies in realizing that since Java is a new language several of the problems which have been identified are likely to be resolved in the future releases of the language, and work is already proceeding in this direction.

9.4 The Device: Java implementation

During the design stage in order to achieve platform independent behaviour it was necessary to divide the communication interface into two layered modules, the Device and the Interface. Both of them have been implemented using the Java programming language, but they have been maintained as separate modules communicating using sockets. The main reason for this decision relies on the fact that since Java cannot access the hardware, in a real application the decision to develop the Device using C or C++ would be an option, and sockets provide the only interprocess communication which Java implements.

All the machine dependent code was confined within the VMD class which had been implemented in a class called `VMD`. By declaring all the variables which characterize the VMD as `static`, any class within the Device can access the VMD

referring to them as VMD.variable. Similarly every class can act on the VMD using methods which have been defined static by calling them with the notation VMD.method(). In this sense VMD acts as a global variable within the Device module. The structure of the VMD class follows:

```
public class VMD extends Thread implements Fms_constants, Device_constants,
                                                    Message_codes
{
  static String model_name = new String(DEVICE_NAME);
  static String manufacturer_name = new String(MANUFACTURER_NAME);
  static float software_version = SOFTWARE_VERSION;
  static String recognition_sys = new String(RECOGNITION_SYSTEM);
  static InetAddress local_host;
  static String db_hostname;
  static String host_name;
  static int logical_status;
  static int physical_status;
  static Vector incoming_job_list;   //list of incoming jobs
  static Vector recent_jobs;         //cache with recent jobs
  static Vector variable_list;       //list of variables
  static Vector event_list;          //list of events
}
```

The information which specifies the identity of the device and its properties are stored in this class. The status variables are used to give an account of the status of the system. physical_status reflects the physical status of the system and can take the values of:

- INITIALIZING;
- IDLE;
- ON_LINE_TEACHING;
- CALIBRATING;
- OPERATING;
- INOPERABLE;
- NEED_COMMISSIONING.

According to the value of physical_status the logical_status may be set to;

- STATE_CHANGES_ALLOWED;
- NO_STATE_CHANGES_ALLOWED.

which indicates if external hosts can act on the internal variables of the Device. Four Vectors are used for handling the incoming jobs, the recent job buffer, the list of variables and events. The structure of the VMD may be suitable for characterizing a large range of devices, but the last two vectors are specifically device dependent. The code segment which is used to control the underlying device is included in the thread called ControlSystem which acts in connection with the VMD class. At start up time the physical_status variable is set to the value INITIALIZING and the logical_status is set to NO_STATE_CHANGES_ALLOWED. Then the variables related with the identity of the system are filled. The name of the host where the database is located is read from a file, thus providing the possibility of changing it without altering the code. All the vectors of the VMD are then initialized. When the device has finished all this activity physical_status is set to IDLE and logical_status to STATE_CHANGES_ALLOWED. This indicates that the device is ready to perform its activities. At this point the ControlSystem thread is started and the normal operations of the device can begin. The code of this thread is completely device dependent.

The whole communication process is based on instances of the class VMD_message whose instances represent messages at the device level. The structure of this class follows:

```
public class VMD_message implements Message_codes, Fms_constants
{
  public int message_type;
  public int data_length;
  public byte[] buffer;
  public byte message_group;
  public String server;
}
```

It completely defines the messages and it is used both for incoming and outgoing messages.

The communication modules allow the Device to be connected with the external world. They have been written to be completely device independent. While the transmission of a message is achieved calling the method `send_message` associated with the `VMD_message` class, the communication module in charge of receiving messages, the `VMD_server`, is implemented using a set of threads. The possibility of running more than one thread with a different priority listening to different ports allows the system to prioritize messages. During the initial stages of the design of the system the idea of using UDP/IP for general messages and event messages and TCP/IP for program messages was proposed. However during the implementation of the UDP code the impossibility of using this protocol for industrial application became apparent. While TCP has built in handshaking procedures which guaranteed that the receiver is ready to accept the incoming message, this is not true for UDP. This protocol just sends the message, and it does not verify if the recipient is willing to receive it. If the receiver is busy processing a message when a new message arrives this is lost. It is possible to implement handshaking and retransmission procedures on top of UDP, but the main risk of this activity is to "reinvent" the TCP protocol. The implementation of the option of making it possible to fragment UDP packets would have increased considerably the code of the system and the possibility of packet losses. In addition the support offered by Java to the TCP protocol is much better then the support for UDP. For these reasons the implementation of the full communication system has been based on the TCP protocol and UDP has been discarded.

When the Device starts up three instances of the `VMD_server` class are created. Each of the instances starts a thread which listens to a specific port with a different priority. They wait until they receive a message from the Interface. When the request for connection for a new message is detected a new instance of the class `Connection_VMD_TCP` is created and a new thread started. This services the incoming message while the `VMD_server` instance returns to listen the assigned port. This means that more than one message can be accepted to be processed at the

same time. The Connection_VMD_TCP thread retrieves the incoming message from the socket storing it in a new instance of the VMD_message class. Then the message is passed to the VMD calling the static method Device.process_message(VMD_message). This method is organized as a case statement which lists the possible associated message codes and the relevant program segment. Different types of messages require different actions. The reply message, if needed, is returned within the processed instance of the VMD_message class, and can be forwarded to the Interface. The operation of sending a message is done by calling the method send_message() associated to the VMD_message class providing the message type, the host to be contacted and the body of the message. This method decides which port to send the message to, connects with the appropriate socket, builds up the message and sends it to the Interface. Then, if necessary, it waits for the reply message.

The Poller is implemented as a thread, and acts in close connection with the VMD class. It performs the task of updating the internal variables within the VMD, and of providing information related to the VMD to the other devices by means of periodically generated messages. When the system starts the first activity of the poller is to read the polling file which contains the list of periodical messages to be executed. For each entry, the action to be carried out (the message code), the host to be contacted, the time interval between two consecutive actions, and eventual additional information, are present. This fact allows the code of the Poller to be device independent since the operations related to the activity of one node are not hardcoded inside the program itself. In addition it is possible to rapidly modify the behaviour of the system, adding, deleting or modifying the entries, without the need of recompiling the code. Since the Poller is supposed to provide periodical information only messages which belong to the class of general messages are meaningful entries in the polling file. Once read, each entry is inserted in a new instance of the class PollingItem which is shown below;

```
class PollingItem implements Message_codes
{
  long polling_time;          //interval of time between two execution
                              //of the defined action
  String host;                //host involved in the operation
```

```
  int action;                  //type of action to perform

  OutputArrayMessage  action_text;
                               //text of the action to perform
}
```

A time tag, indicating when the action has to be performed next time, is then attached to each instance of `PollingItem` which is then inserted in a priority queue. This data structure allows the entries to be ordered in a such a way that the one which has the smallest time stamp (the first one which needs to be serviced) always occupies the first position of list. When the initialization procedures have been terminated, the Poller enters in an infinite loop. Here the first element of the priority queue is retrieved, and the difference between the current time and the service due time is calculated. If this value is negative the action is immediately performed and a warning message generated, otherwise the thread is placed to sleep for the appropriated interval of time before sending the message. Then a new time stamp for the `PollingItem` is calculated before inserting it again in the priority queue. The Poller follows two different procedures which involve the generation of the appropriate message for servicing the entries of the priority queue. For operations which require external information (such as `STATUS` or `READ`), first the appropriate message is generated calling the `send_message()` method, then the body of the reply, is forwarded to the `VMD` for processing. For actions involving internal data (such as `UNSOLICITED_STATUS` and `WRITE`) first it is necessary to query the VMD. This is done by the Poller issuing a virtual request to the VMD, processing the answer and then sending it. For example if it is necessary to send a `UNSOLICITED_STATUS` message to another device, first of all the Poller interrogates the VMD generating a `STATUS_req` message. The body of the `STATUS_ans` reply is then included into a `UNSOLICITED_STATUS` message which is then sent using the `send_message()` method. The periodical activities performed by the poller could be subject to real-time constraints. As Java does not provide support for real-time, this means that the Poller might suffer the problem of missed deadlines.

9.5 The Interface: Java implementation

The interface act as a filter between the Device and the network. It is used to encode messages from local format to network format and decode them. The use of Java in this environment is ideal since it provides all the features necessary for implementing the networking code.

All the messages handled by the Interface are stored as instances of the `Message` class. The structure of the class follows;

```
public class Message
{
  byte[] add_receiver = new byte[4];
  byte[] add_sender = new byte[4];

  int message_type;

  private static int this_ref_number=-1;
  int ref_number;

  int data_length;

  byte[] buffer;
}
```

The Interface is composed of two separate entities, the Interface Client which listens for messages from the Device and the Interface Server which listens for messages from the external world. The structure of the two modules is similar to the one for the VMD server since each of them runs three threads with different priorities which listens for different types of messages. When a new incoming message is present a new thread for servicing the message is started. Then the main thread returns to listen to the assigned port. In the Interface Client first the name of the client is received and the associated Internet address retrieved. Then the rest of the message is read and included in a instance of `Message`. A progressive reference number is associated to every message. The writing operation performs the encoding of the message from the local representation to the global Java representation using write methods associated with the `DataOutputStream`

class defined in Java. Once the message is completely copied in the buffered output stream, the buffer is flushed, and the message is sent along the network. If a reply message is expected the client waits for the reply. At the other end the Interface Server waits for incoming messages. When a message arrives it creates a new thread which handles the communication. The header and the body of the message are retrieved, decoded and stored in an instance of the class `Message`. During this operation the local address and the destination address are matched and the message code verified. Then the message is sent to the VMD Server in the form of VMD Message. If a reply is required the Interface Server waits for it. When it is produced it reads the incoming VMD Message, processes and sends it back to the client station. At the other end the message is received, and the addresses of the stations involved, the message code and the reference number are checked to verify the consistency of the reply. Then it is decoded and sent back to the Device.

9.6 The production control system: Java implementation

The production control system supervises the whole production activity of the plant sending messages to all the connected nodes in order to ensure the correct behaviour of the system. The system is composed of several interacting modules; the communication subsystem, the poller, the monitor, the scheduler and the dispatcher. It extensively makes use of the communication system, since it needs to receive periodic updates of the overall status and send messages to the connected nodes. All the periodic communications are handled by the poller. Since the structure of the communication modules and the one of the Poller are exactly the same described for the Device, they will not be presented again. As in the case of the Device all the device dependent code has been limited to a single module.

The monitor is the centre of the production control system. It stores an image of the workcell. Here all the information related to the connected devices, the

components and the scheduled jobs is stored. The structure of the monitor resembles the VMD. Since the data held here may be accessed by more than one thread simultaneously synchronization procedures have been necessary. In order to allow the reuse of the modules which have been designed for the VMD the Java class which implements the monitor has been called VMD. Since only one instance of the class monitor is necessary all the variables have been defined as static, and no constructor to the class has been provided. In addition to the variables already encountered in the VMD class defined in the Device, other variables which are needed by the monitor to keep a record of the system are present. They deal with the input and output bays used by the workcell, the list of the machines included in the workcell, the job list, and the schedule. The code of the class follows:

```
public class VMD implements Fms_constants, Monitor_constants,
                            Message_codes, ControlSystem_constants
{
  static String model_name = new String(DEVICE_NAME);
  static String manufacturer_name = new String(MANUFACTURER_NAME);
  static float software_version = SOFTWARE_VERSION;
  static InetAddress local_host;
  static String host_name;
  static int logical_status;
  static int physical_status;
  static Vector variable_list;
  static Vector event_list;

  //defines the inpout and output trays used by the workcell
  public static Tray in_tray = new Tray(IN_TRAY,IN_TRAY_SLOTS);
  public static Tray out_tray = new Tray(OUT_TRAY, OUT_TRAY_SLOTS);

  //defines the list of the machines in the workcell it is made
  public static Vector work_cell = new Vector(NUMBER_MACHINES);

  //defines the job list
  public static Vector job_list = new Vector();
  //semaphore on the scheduler
  public static boolean sched_ready;
  //defines the actual schedule
  public static Vector sched = new Vector();

}
```

Each tray is represented with an instance of the Tray class which defined the

tray name, its number of slots and includes an array of instances of the class `Slot`. These store the name and the quantity of the component held. At the start up of the system the status of the input tray is requested. The function `filltray()` is called and the contents and the amount of each input slot are requested. When a component is inserted the system immediately makes a query to the database system to verify the existence of the component name. The list of the machines in the workcell is stored in a Vector structure. This allows simple modification of the code is a new device is inserted in the workcell. It is initialized at start up time. The Vector contains instances of the class `Device_data`. The template of the class is:

```
public class Device_data implements ControlSystem_constants
{
  public String name;
  public String model_type;
  public int logical_status;
  public int physical_status;
  public String model_name;
  public String manufacturer_name;
  public float software_version;
  public float operating_time;
}
```

Each instance of the `Device_data` class has to provide the information necessary to define the devices present in the workcell and their status. The list of jobs is stored in a Vector structure since the number of jobs is not known in advance and offers better flexibility and access function than an array structure. For a similar reason the schedule itself is organized using a Vector. The Monitor is updated by calling the `VMD.process_message()` method, called in this way for compatibility reasons with the software developed for the Device. It consists of a long case statement. Depending on the code of the incoming message the appropriate set of actions are performed.

While the monitor is a kind of data repository where dynamic information is stored, the scheduler, implemented in the class `Scheduler`, and the dispatcher, implemented in the `ControlSystem` class, are implemented as threads. The scheduler

is dormant for most of the time, and wakes up when new component to be processed is inserted. At start time when the slots of the input tray are filled it wakes up and generates the schedule. The class `Scheduler` works in close conjunction with the Database system from where it gets all the relevant information about the component to be processed. It creates the list of the jobs, analyzing the component contained in the input tray, then it generates the schedule. The entries of job list are instances of the `Job` class. The structure of the class follows:

```
class Job implements ControlSystem_constants
{
  String comp_name;              //the name of the output component
  static int seq_number = 0;     //job number
  int output_tray_slot;          //output slot
  int quantity;                  //pieces produced
  int to_go;                     //pieces to produce
  float[] timing= new float[NUMBER_MACHINES];
                                 //timing information
  int number_jobs;               //number of components to process
  Slot[]  jobs_order;            //list of components
  Vector  job_times;             //timing information generated by
                                 //the dispatcher
}
```

As it is possible to see the class is quite complex. The scheduler analyzes in turn all the entries of the input tray. For each component present it interrogates the database to see if it is a single component or if it is a part of something more complex. If the component is single a new instance of `Job` is created and, the processing times retrieved from the database and inserted in the `timing` array. If the component is part of a multiple component, the `job_list` is scanned to see if the resulting component already has an entry. If so the the component is added in the relevant position in the `jobs_order` array. Otherwise a new entry is created, the timing information retrieved from the database and the component is inserted in the `jobs_order` array. When the input tray has been completely scanned the `job_list` is processed to see if all the components necessary to produce multiple components are present in the input tray, and their numerical consistency is calculated. The jobs that can be processed are then inserted in the `sched` vector which

is passed to the scheduling algorithm that reorders the list in the appropriate way. During these operations the boolean value `sched_ready` has been set to *false* in order to stop the dispatcher having access to the elements of the scheduling list. When the schedule is finally ready `sched_ready` is set to *true*, and the change is notified with a `notifyAll()` command. Then the scheduler resumes its dormant state waiting for new components in the input tray.

The dispatcher, implemented in the `ControlSystem` class, once the schedule is ready retrieves its first element of the scheduling list. Then the `job_times` information, which determines the precise time when the components have to go in production, is calculated. Then if all the devices are active a message containing the data relevant to the next component to be processed is sent to all the devices connected in the production line using a `WRITE` command.

9.7 Java in distributed manufacturing systems

During the development of the system it has been possible to assess the key benefits and limitations of Java for the implementation of a communication and support system for industrial automation, and more generally on the possibility of using this language in other industrial applications. Some of the features of the language have been highlighted in section 9.3, but during the implementation of the system other advantages of the language have become evident.

The language has proved to be easy to learn starting from a C background. Its simplicity allowed a rapid development cycle of the system. The absence of pointers dramatically reduced the number of bugs and errors during the development of the code. Range checking on array access immediately identified irregular operations enabling problems to be fixed at the source. Automatic garbage collection eliminated the need to deallocate dead objects and eliminated the danger of destroying the ones in use. The support for threading and networking has provided the means for implementing complex interacting modules in a simple way. The absence of integrated development tools and debuggers has not been a relevant problem during the implementation phase.

The presence of utilities classes among the standard libraries of Java have been extremely valuable during the development of the system. It has been possible to used some data structures directly without the need to develop code for the application. The class `Vector` provides the necessary features for implementing different types of lists required inside the Device and the Control System modules. The class `Hashtable` has been used in for storing and retrieving the features associated with the message codes.

The language offers a good support for the TCP protocol. When a TCP socket is used it is possible to retrieve its input and output streams. This permits applications in several types of I/O abstract classes giving a high level of control on the input and output through the socket. For example in the developed application every socket output stream is first connected to a buffered output stream and then wrapped with a `DataOutputStream`. The relevant code follows:

```
//creates the buffred output stream
buffer_forward =
       new BufferedOutputStream(this.forward_socket.getOutputStream());
//wraps it in another stream
forward_out = new DataOutputStream(this.buffer_forward);
```

All the methods associated with the `DataOutputStream` class can be called by the `forward_out` object and the ones associated with the `BufferedOutputStream` class by the `buffer_forward` object. The buffered stream prevents the message being fragmented into small packets as could happen if data is directly written to the output stream connected to a socket. The `DataOutputStream` class offers a full range of methods which allow all the basic data types, strings and arrays to be written in an output stream.

The possibility of reading and writing the basic data types into the TCP sockets offered by the methods of the `DataOutputStream` and the `DataInputStream` classes provide Java with the means of transmitting data encoded in a platform independent way. In the implemented system the main activity of the Interface is to transform the data from the local format to a standard "Java" format to be transmitted along the network using these methods. This feature of the language

makes unnecessary the definition of encoding rules for simple data types which have to be transferred along the network.

At the level of the Device, the method used for sending messages requires as an input the body of the message in the form of a an array of bytes. For this reason methods for inserting and retrieving basic data types into byte arrays have been developed at the Device level. The main reason for this decision is the fact that it was assumed that the code of the Device could be written in any language, and therefore it would not be possible to suppose the presence of the Java support classes for reading and writing basic types into streams. In addition one of the initial fields of every message reports its length. If the length of the message is not known in advance it is not possible to write the data directly into the output stream.

Java does not allow global constants which are visible to all the classes, and they have always to belong to a class, therefore the notation for addressing them must be `ClassName.CONSTANT`. This is sometimes a tedious notation which decreases the code readability. A solution to this problem is to define the constants inside an `Interface` and making the class that needs to use those constants implementing the Interface. Then it is possible to use the name directly, assuming there are no name clashes. Similarly it is possible to define global variables inside a class and declare them `static`. Such variables must be addressed with the `ClassName.VarName` construct. Alternatively it is possible to define the classes that need access to these variables subclasses of the class defining the variable.

The possibility of defining global constants inside an Interface allows the implement of *enumerated* types which are not directly defined in Java. The result looks like this;

```
public interface Message_codes
{

  //messages for the VMD support

  public final static int STATUS_req = 0x0004;  //bin: 0100
  public final static int STATUS_ans = 0x0005;  //bin: 0101
  public final static int STATUS_err = 0x0006;  //bin: 0110
```

```
}
```

The result is not as concise as it would be in C and there is the problem that it is possible to pass any value with compatible type (in this case any `int`) to methods which expect as arguments one of the values defined in the body of the enumeration statement. For this reason it is always necessary to include lines of code which check the validity of the passed values. In addition enumeral values can be duplicated accidently. A more sophisticated solution which solves the type safety problem has been suggested by [JS97] making use of the `instanceOf` type-safe dynamic cast. However in this case the code is even more complex and may appear less clear to the reader since one class has to be defined for each enumerated type.

Java does not define pointers to methods. Their use could have found a useful application in the communication system. At the receipt of a message its code would have been examined and the appropriate action defined using a pointer to the appropriate method. The same result has been achieved with a long `case` statement, but the code implemented using pointers to methods would have been more elegant. A solution to this problem have been proposed by [Lea96], but it is of complex implementation.

9.8 Conclusion

The implementation stage permitted the transformation of the classes and relationships developed during the design stage in a working system. Since both the database and the programming language were object oriented the transfer process from the design to the implementation was performed without major problems since the structure defined at the design stage had been promptly and correctly mapped into the applications. Java has proved to be a valuable programming language for the development of complex network based applications, and it has shown that is has some important strengths for use in the development of manufacturing control

systems; it provides major advantages in its portability and machine independence. However, the current state of the language is such that some important features such as direct access to the underlying hardware are lacking. From the analysis carried out it is possible to conclude that Java has potential even in the area of real-time applications, but it cannot be considered as the language which is going to resolve all the problems related with real-time programming. In particular it is important to be able to distinguish Java the language from its other parts, since there is some confusion among the general public. As a language, Java has great advantages of C and C++ in terms of safety and simplicity, even if it lacks of some useful language features, such as enumerations, and pointers to methods. If Java is to realize its potential in this field, appropriate additions to the language will need to be made, and further work on language development is required. This needs to be accomplished without the loss of the portability which is fundamental to the system. A step is this direction has already been made as Sun Microsystems has announced that it is working on the "Java Embedded APIs" : a variation of Java for embedded devices that are incapable of supporting the full Java Core API. It is likely that an addition to the Java Virtual Machine might provide an ordered and fully defined method of pointer implementation. In the next chapter the performance of the system and of the Java implementation will be assessed.

Chapter 10

System Analysis

10.1 Introduction

The implementation of the system designed with the OMT model in Java with the support of the Postgres95 database management system has proved the validity of the use of the language for the implementation of manufacturing applications. However another aspect that must be considered is the performance issue.

Performance is important in many application and becomes vital if systems have to deal with time and deadlines. Java is usually an interpreted language which means that the so called "compilation" of a Java program (using for example the `javac` command) does not produce an executable file (such as the compilation of a C program), but it creates Bytecode. A Java Virtual Machine (a program written in ANSI C) is needed to interpret the Bytecode and execute the application on the local platform. This produces a loss in performance of Java programs if compared with equivalent applications written in C. This has been extensively reported and has been openly admitted by Sun in their Java Whitepaper [GM95]. However this is rapidly changing and recently *Just in Time* (JIT) compilers, which perform a code compilation at run time translating the Java Bytecode in the host machine's native code, and Java compilers, which produce platform native code, have been introduced in order to speed up performance of Java applications. Since no data related to the performance of the Java on Linux was available, a series of tests

were performed. Interpreted Java, just in time compiled Java, compiled Java and C and C++ were compared. The results obtained are reported in appendix D. From the data gathered it is possible to see that Java, in any flavour, is always less efficient then a program written in C or C++. Only in few cases the performance is comparable. Allocation of new objects in memory is always a particular expensive operation in comparison with C and C++. Therefore limiting the creation of new objects in the code is a way to improve performance of Java applications. However, since the Byte magazine benchmarks test several features of the language at the same time, they are the ones which give the best indication of the performance of Java. From these tests it is possible to see that interpreted Java applications on Linux run from 20 to 40 times slower than equivalent programs written in C or C++. These numbers improve if the applications are compiled just in time (3-15 times slower) or compiled in native code (1.5-7 time slower). Due to the considerable improvement in performance offered by the non-interpreted versions of Java, it is possible to foresee their wider acceptance in future.

In section 10.2 the details of the platforms and software versions used during the tests are given. Section 10.3 presents the results of a set of experiments performed to assess the performance of the TCP network implementation of Java. In section 10.4 the performance of the database is assessed. Section 10.5 presents the results of the experiments carried out to assess the response times of the system to selected messages. Section 10.6 presents the results of the performance of the cell in particular conditions. Finally the results of the simulation of a manufacturing cell controlled by the implemented support system are reported.

10.2 System configuration

The software developed for the communication and control of the manufacturing cell has been tested on a series of computers which simulated devices present on the cell. A Pentium 75 MHz, 16 Mbytes RAM, 2.2 Gbytes HD, running Linux 2.0.25, hosted the workcell monitoring system and the database system. For simulating the other devices a series of four Intel 80486 33 MHz platforms, with 8 Mbytes of

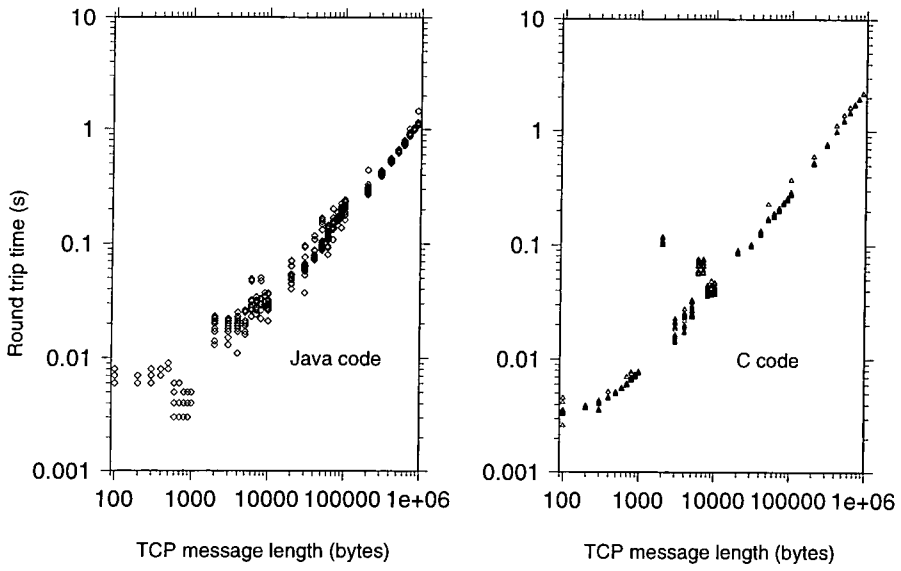


Figure 10.1: Performance of the same application for measuring TCP round trip transmission times written in Java and C on an Intel 486 platform running Linux 2.0.29

RAM, 1.2 Gbytes HD running Linux 2.0.29, were used.

Different series of tests have been performed using the interpreted and just in time compiled version of the program. Unfortunately it has not been possible to evaluate the performance of the the compiled version of the program since Toba, the Java compiler, did not have thread support. However it has been possible to use compiled code for the test of the performance of the database since this did not use threads.

The `java` interpreter from the JDK 1.0.2 port for Linux, the `kaffe` JIT compiler version 0.84, and the `toba` Java compiler version 1.0.b6 have been used during the tests.

10.3 Java network performance

The first activity in order to assess the performance of Java consisted in testing its network capabilities.

The code which has been used to measure the performance of the TCP protocol, whose results have been reported in chapter 3, has been translated into Java. The test has been carried out on the same pair of Intel 486 platforms running Linux

2.0.29 used in one of the tests done with the C code. The programs have been compiled with the `javac` compiler and executed using the JavaVM developed for Linux supporting Java 1.0.2. The results of the test are reported in figure 10.1. The performance of the C code is reported for comparison. The first impression obtained from the Java graph is the large number of scattered point in comparison to the C implementation which presents a more consistent behaviour. Secondly it is also possible to notice that the round trip time of small packets assume only discrete values. This is caused by the limitation of the Java timer which cannot measure time intervals which are smaller than one millisecond. It is possible to notice that the transmission time is almost constant for packets up to 500 bytes and then it suddenly drops. It is suggested that this behaviour is connected with something similar to the Nagle algorithm, which waits the expiration of a timer before sending small packets. This is due to the fact that longer packets are more efficient to transmit. The most astonishing feature of Java consists in the fact that above 600 bytes, it achieves round trip times which are smaller than those obtained with the C code. This can be explained by the fact that the test timed the performance of the system calls to TCP services which are passed by the application, through the JavaVM, to the system, so that only a limited amount of Java code is involved in the operation. In addition it is likely that the JavaVM has been optimized in order to achieve better performance during TCP transactions. This would explain the absence of the of irregularities present in the C code which are associated with the Nagle algorithm. The scattering of the points on the graphs is likely to be caused by the JavaVM, and in particular they may be associated with the activity of the garbage collector which becomes active to reclaim memory.

These results allow us to conclude that the Java code has a very good network performance even if the results are subject to time variations. They are in accordance with the tests carried out on the Java I/O system reported in appendix D, which show that the Java performance is comparable with the one of applications written in C and C++.

query	mean (s)	stdev	tuples
SELECT * FROM bad_shape	0.046	0.0046	1
SELECT * FROM stitching_data	0.050	0.0027	2
SELECT * FROM processing_time	0.038	0.0007	3
SELECT * FROM device	0.064	0.0062	4
SELECT device.model FROM device WHERE device.name = 'gemini'	0.032	0.0046	1
SELECT device.model FROM device WHERE device.name = 'scorpio'	0.032	0.0057	1
SELECT processing_time.processing_time FROM processing_time WHERE processing_time.device_model = 'autostitcher'	0.034	0.0016	2
SELECT processing_time.processing_time FROM processing_time WHERE processing_time.device_model = 'autostitcher' AND processing_time.component_name = 'compA'	0.040	0.0027	1
SELECT component_position.part_of, component_position.position FROM component_position WHERE component_position.name = 'compB' AND component_position.joining_device = 'placing_robot'	0.046	0.0161	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	0.046	0.0114	1
SELECT stitching_data.data_lenght FROM stitching_data WHERE stitching_data.comp_name = 'compA'	0.032	0.0021	1

Table 10.1: Database query performance times obtained with the interpreted version of the test program. The interval between the queries is 1 second.

10.4 Database performance

Since an important part of the system activity involves interaction with the database, an evaluation of the query times were performed. The Postgres95 version 2.0 was used. All the queries were obtained using a program written in Java using the Java-Postgres95 0.2 programming interface to query the database. This set of library routines allows client programs to pass queries to the Postgres95 DBMS and

to receive the results. About 70 entries were stored in the database in the different tables. Queries of different complexity have been formulated in order to ascertain whether a relationship between complexity and time was present. They were built taking as examples possible queries made by the devices to the database. All the queries were loaded into a list, and repeated 20 times cyclically. The time between queries was one second.

Some of the results gathered during the tests with the interpreted Java are reported below in table 10.1. These show that the complexity of the statement apparently does not influence the time required for performing the query. Similarly the number of *tuples* (instances) retrieved does not seem to affect the query time. The results show that the time required for a query is on average between 0.03 to 0.05 seconds with only one case of a query whose average is 0.06 seconds. The value of the standard deviation indicates that the results are usually consistent. The same test program was executed using the Kaffe JIT compiler, and transformed in native code using the Toba compiler. The interval between the queries was set to 0.1 seconds. A selection of the results is presented in table 10.2. As it is possible to see, the performance of the three versions of the tests produce comparable results with a slightly better performance for the compiled version. This indicates that the largest amount of time is spent within the DBMS module and differences in the speed of the Java application do not influence the results.

Postgres95 users have reported on the Usenet that the query times increase with the number of entries in the database. However if the tables are stored in the database using a *B-tree* structure (by using the appropriate command at the creation of the table) the query time becomes almost independent of the number of entries.

During the experiments it has been discovered that the frequency of the queries influences the performance of the database. A comparison of the execution times of the same query with different time intervals between queries is reported in table 10.3. If the query function to the database was placed inside a continuous for loop, so when a query returned it was immediately followed by another one, the average query time was from 0.09 to 0.13 seconds. The performance of the database im-

query	method	mean (s)	stdev	tuples
SELECT * FROM bad_shape	java	0.044	0.0019	1
SELECT * FROM bad_shape	kaffe	0.037	0.0008	1
SELECT * FROM bad_shape	toba	0.030	0.0010	1
SELECT device.model FROM device WHERE device.name = 'gemini'	java	0.035	0.0127	1
SELECT device.model FROM device WHERE device.name = 'gemini'	kaffe	0.036	0.0233	1
SELECT device.model FROM device WHERE device.name = 'gemini'	toba	0.030	0.0074	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	java	0.046	0.0074	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	kaffe	0.041	0.0021	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	toba	0.042	0.0040	1
SELECT stitching_data.data_lenght FROM stitching_data WHERE stitching_data.comp_name = 'compA'	java	0.031	0.0015	1
SELECT stitching_data.data_lenght FROM stitching_data WHERE stitching_data.comp_name = 'compA'	kaffe	0.029	0.0044	1
SELECT stitching_data.data_lenght FROM stitching_data WHERE stitching_data.comp_name = 'compA'	toba	0.030	0.0077	1

Table 10.2: Database query performance times with a an interpreted (java) JIT compiled (kaffe) and compiled (toba) version of the test program. The interval between the queries is 0.1 seconds.

proved if a delay between the queries was set. With delays of 0.1 seconds between queries the performance was the same as registered for for intervals of 1 second, but with a higher standard deviation. This behaviour is probably related to the way in which the DBMS system handles the queries through TCP connections.

query	interval (s)	mean (s)	stdev	tuples
SELECT * FROM processing_time	none	0.097	0.0129	3
SELECT * FROM processing_time	0.010	0.101	0.0097	3
SELECT * FROM processing_time	0.026	0.081	0.0099	3
SELECT * FROM processing_time	0.104	0.049	0.0113	3
SELECT * FROM processing_time	1.048	0.046	0.0046	3
SELECT device.model FROM device WHERE device.name = 'gemini'	none	0.111	0.0159	1
SELECT device.model FROM device WHERE device.name = 'gemini'	0.010	0.096	0.0097	1
SELECT device.model FROM device WHERE device.name = 'gemini'	0.026	0.045	0.0090	1
SELECT device.model FROM device WHERE device.name = 'gemini'	0.104	0.035	0.0127	1
SELECT device.model FROM device WHERE device.name = 'gemini'	1.048	0.032	0.0046	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	none	0.104	0.0125	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	0.010	0.104	0.0133	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	0.026	0.074	0.0098	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	0.104	0.046	0.0074	1
SELECT component.tot_components FROM component* WHERE component.name = 'compA'	1.048	0.046	0.084	1

Table 10.3: Database query performance times in relation to the interval between queries using the interpreted Java test program

10.5 Message timings

The manufacturing control system relies on messages for the exchange of information between the connected entities. Section 10.3 has highlighted the performance of the TCP function calls of Java. In this section the performance of complete communication system is assessed.

Each message sent by the Device must traverse the Interface before being sent

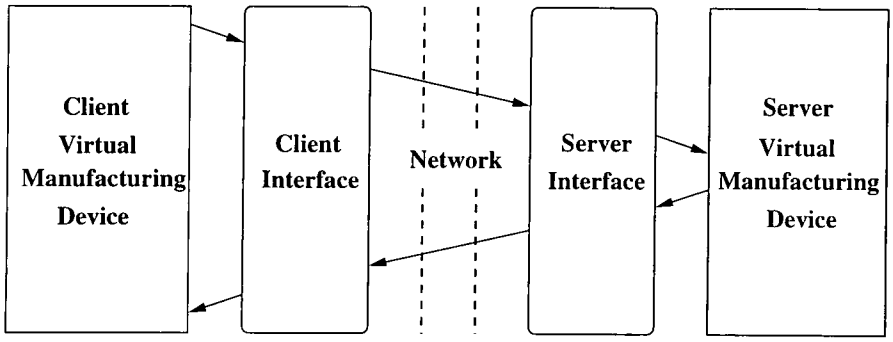


Figure 10.2: Message passing between the different layers implemented in the manufacturing communication system.

along the network. On the server side the message is received by the interface and forwarded to the VMD Server which acts properly on the Device. If an answer is needed the VMD generates it and sends it back to the client passing through the Interfaces. A scheme representing the message passing is shown in figure 10.2.

Each message implemented in the transmission protocol belongs to one of the three defined categories: general messages, program messages, and event messages. During the test it was only possible to collect timing data related to messages which require acknowledgment, due to problems associated with synchronizing the internal timers of the computers.

The communication time required for sending a message between two 80486 machines, from a 80486 to a Pentium, and the opposite were measured. This was necessary in order to assess the difference of time required by the transaction. Due to the similarity of the messages only a sub-set has been tested. The timer starts when the `send_message()` method is called and stops then it returns. Each test was repeated 20 times in order to have statistical significance. Tests with three different message types were performed; `IDENTIFY` and `STATUS` belonging to the set of general messages, and `STOP` belonging to the event messages. The results of the tests obtained when the interpreted version of the developed application was run are presented in table 10.4. It is possible to see that the times are influenced by the platform used, but all of them are in the order of 0.1 seconds. The number of operations required to produce the reply message were comparable. However an increase in the transmission time is to be expected if more complex operations, such as database accesses, are to be performed on the server platform before the

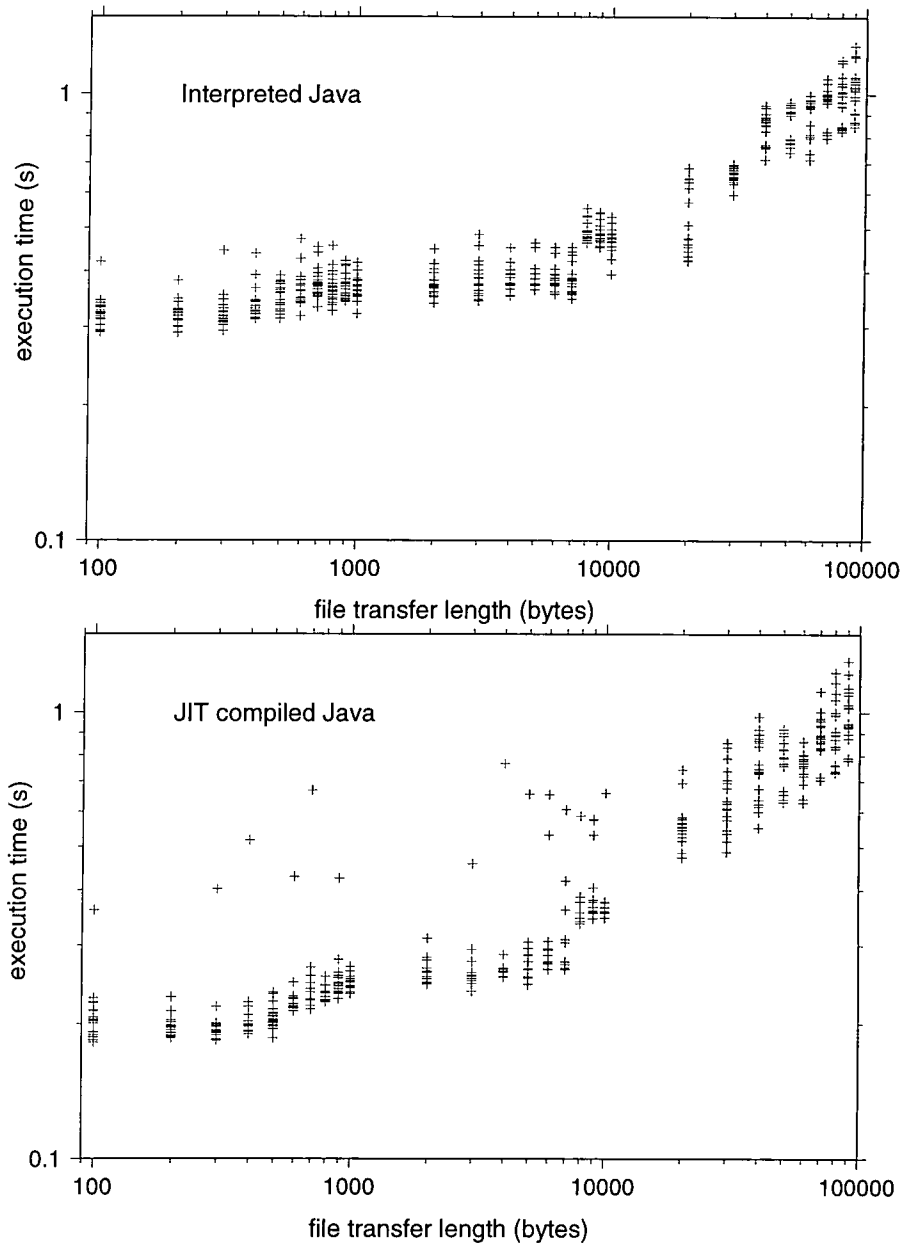


Figure 10.3: Time required for uploading a data file in relation with its length

reply message could be sent. The same tests were performed running the system using `kaffe`, so obtaining a JIT compiled version of the application. The results of these second round of tests are shown in table 10.5. It is possible to see that in this case the the response time of the selected messages is halved. This is in accordance with the results obtained from the benchmarks.

Other tests were carried out to record the response time to the set of messages associated with the operation of downloading a data program from the database. Data files from 100 to 90000 bytes were transferred in order to assess the relation-

ship between time and file length. The interval between each download request was 0.1 seconds. 20 repetitions for each data length request were performed. The data file request procedure consists of two consecutive messages, `INITIATE_UPLOAD_FILE` and `UPLOAD_FILE`. The first one queries if the requested program is present in the database and its length. The second message triggers the downloading of the program itself. These operations involve two database queries (one for each message). The process of retrieving the data was simulated by performing two generic queries. Figure 10.3 shows the results the experiments in the cases of interpreted and JIT compiled execution of test. It is possible to see that in the case of the interpreted version of the application the transfer times are between 0.3 and 0.4 seconds for message requests between 100 and 7000 bytes. Then the transmission time increases. For messages of 100 bytes the mean transmission time is 0.32 seconds which is consistent with the time required for the transmission of 2 messages (0.2 seconds) and the two queries to the database (which had been timed and took about 0.11 seconds). It is possible to see that the time required for the transmission of a data file whose length is between 100 and 7000 bytes is almost independent from the length of the data file. After that there is a discontinuity between 7000 and 8000 which is probably caused by the exhaustion of the allocated TCP buffer space (8000 bytes). The graph associated with the measurements taken with the JIT compiled version of the application show a better performance for small file transmissions. However for data files larger than 10000 bytes the difference between the two is not evident. For small messages the spread of the measurements is smaller in comparison with the one present in the interpreted version of the test. It is possible to see that sometimes the time required for the transmission is much larger than the average. The discontinuity in performance between 7000 and 8000 bytes is evident in this graph as well.

A further test consisted in measuring the overhead introduced by the the Interface during the transmission procedures. The client and server part of the Interface was analyzed. The following measures were taken:

- on the client side the time interval from when the Device sends the message to the network to when the Interface performs the same action (the time

between the two `buffer.flush()` operations) was recorded;

- on the server side the interval between when the message is accepted by the Interface and the same action is performed by the Device (the time between the two `accept()` operations) was measured.

The times required by a reply message to traverse backwards the two Interfaces were also measured.

- on the server side the time between when the message is sent by the Device and when it is sent by the Interface (`buffer.flush()` operations) was taken;
- on the client side the interval between the first read on the Interface and the first read of the reply message on the VMD was taken.

The transmissions were repeated 20 times and the average calculated. The results are shown in table 10.6. When the Pentium was used as master and the 80486 as slave with a `IDENTIFY` message, it took on average 19 ms to traverse the client side of the Interface, and 27 ms were spent on the server side before reaching the Device. The reply part of the message took 12 ms to go through the server side, and 5 ms to traverse the client side. In a second test the computers were interchanged. In this new scenario the time required to traverse the client Interface on the 80486 was 45 ms, and only 10 to go through the server side. The reply message spent on average 3 ms on the server side of the Interface and 11 on the client side. These results show how the performance of Java is influenced by the speed of the platform where it is executed. Other measure revealed that half of the time spent by a message on the server side is associated with the Interface procedures of validation, coding and decoding of the messages.

During the tests it has only been possible to collect timing data related to messages which require acknowledgment. However from the data acquired during the measurements of the transmission times involved in traversing the Interface it is possible to see that messages spend a larger amount of time in the Interface in their outbound trip. From this data it is possible to deduce that the transmission

message type	client	server	mean	stdev
IDENTIFY	80486	80486	0.141	0.0107
STATUS	80486	80486	0.145	0.0130
STOP	80486	80486	0.138	0.0136
IDENTIFY	80486	Pentium	0.098	0.0090
STATUS	80486	Pentium	0.101	0.0139
STOP	80486	Pentium	0.097	0.0126
IDENTIFY	Pentium	80486	0.099	0.0099
STATUS	Pentium	80486	0.102	0.0124
STOP	Pentium	80486	0.098	0.0124

Table 10.4: Message execution times with interpreted application using `java`

message type	client	server	mean (s)	stdev
IDENTIFY	80486	80486	0.072	0.0018
STATUS	80486	80486	0.077	0.0050
STOP	80486	80486	0.071	0.0033
IDENTIFY	80486	Pentium	0.052	0.0021
STATUS	80486	Pentium	0.051	0.0010
STOP	80486	Pentium	0.051	0.0024
IDENTIFY	Pentium	80486	0.053	0.0044
STATUS	Pentium	80486	0.051	0.0015
STOP	Pentium	80486	0.049	0.0037

Table 10.5: Message execution times with JIT compiled application using `kaffe`.

time for a message which does not require acknowledgment is about the 60-70% of the time required by acknowledged messages to perform a round trip.

10.6 Simulation tests

In the previous set of tests an assessment of the performance of the communication system was carried out. Only two devices were involved in the transactions. In order to simulate a realistic scenario which represents the arrangement of a factory model, the computers were interconnected together in a network. A series of test

message	client	server	Interface client traverse time		Interface server traverse time	
			mean (s)	stdev	mean (s)	stdev
IDENTIFY_req	Pentium	80486	0.019	0.0025	0.027	0.0023
IDENTIFY_ans	Pentium	80486	0.005	0.0008	0.012	0.0016
IDENTIFY_req	80486	Pentium	0.045	0.0058	0.010	0.0013
IDENTIFY_ans	80486	Pentium	0.011	0.0006	0.003	0.0002

Table 10.6: Overhead introduced by the Interface during the transmission process.

method	interval (s)	best		worse		reference
		mean (s)	stdev	mean (s)	stdev	mean (s)
java	2.00	0.097	0.0116	0.115	0.0121	0.098
java	1.00	0.103	0.0151	0.110	0.0150	0.098
java	0.50	0.097	0.0135	0.109	0.0216	0.098
java	0.10	0.108	0.0167	0.113	0.0295	0.098
java	0.05	0.109	0.0239	0.135	0.0267	0.098
java	nil	0.134	0.0389	0.150	0.0364	0.098
kaffe	2.00	0.054	0.0136	0.067	0.0390	0.052
kaffe	1.00	0.054	0.0136	0.059	0.0343	0.052
kaffe	0.50	0.061	0.0444	0.074	0.0186	0.052
kaffe	0.01	0.064	0.0162	0.074	0.0220	0.052
kaffe	0.05	0.065	0.0191	0.075	0.0211	0.052
kaffe	nil	0.070	0.0160	0.075	0.0198	0.052

Table 10.7: System performance under increasing load (decreasing time between messages).

aimed to assess the capabilities and monitoring the behaviour of the communication system in special conditions were performed.

A subsystem common to all the devices is the Poller, the module whose task is to generate periodical queries. These messages enable monitoring the activity of the system and provide information for the correct operation of the plant. Tests have shown that the operations associated with the poller for sending a message do not add a noticeable overhead on the transmission time of messages.

A further experiment consisted in assessing the performance of the system under increasing load. This was done by connecting to an isolated network four 80486

method	message	pri	mean (s)	stdev	reference mean (s)
java	STOP	high	0.098	0.0134	0.097
java	UPLOAD_FILE	normal	0.395	0.0617	0.325
java	IDENTIFY	low	0.183	0.0608	0.098
kaffe	STOP	high	0.105	0.0577	0.051
kaffe	UPLOAD_FILE	normal	0.262	0.0518	0.214
kaffe	IDENTIFY	low	0.109	0.0590	0.052

Table 10.8: System performance under priority race condition with a file of 100 bytes downloaded from the database

db file transfer	message	pri	mean (s)	stdev	reference mean (s)
100	STOP	high	0.098	0.0134	0.097
100	UPLOAD_FILE	normal	0.395	0.0617	0.325
100	IDENTIFY	low	0.183	0.0508	0.098
1000	STOP	high	0.102	0.0130	0.097
1000	UPLOAD_FILE	normal	0.467	0.0541	0.351
1000	IDENTIFY	low	0.191	0.0559	0.098
10000	STOP	high	0.108	0.0134	0.097
10000	UPLOAD_FILE	normal	0.611	0.0702	0.472
10000	IDENTIFY	low	0.192	0.0899	0.098
30000	STOP	high	0.107	0.0146	0.097
30000	UPLOAD_FILE	normal	0.787	0.0958	0.648
30000	IDENTIFY	low	0.184	0.0850	0.098
60000	STOP	high	0.108	0.0296	0.097
60000	UPLOAD_FILE	normal	0.974	0.1086	0.851
60000	IDENTIFY	low	0.171	0.0931	0.098
90000	STOP	high	0.104	0.0331	0.097
90000	UPLOAD_FILE	normal	1.172	0.1354	1.018
90000	IDENTIFY	low	0.161	0.1057	0.098

Table 10.9: System performance under priority race condition with a file of different sizes using interpreted Java

machines acting as clients and requesting a service (`IDENTIFY`) all at the same time to the manufacturing monitoring system running on the Pentium. The time interval between two consecutive requests was set as a parameter of the experiment. Each request was repeated 20 times. The results of these tests are shown in table

10.7. The best and the worst performance among the four clients is reported. Reference times measured when only one machine is accessing the server are displayed for reference. For the interpreted Java version the time between the messages does not affect the response time up to interval times of 0.5 seconds, and there is a difference of about 10% between the reference time and the recorded time. However for messages with smaller cycles the standard deviation, and the average, increase. The results associated with Kaffe show a less defined trend since the mean increases with the decreasing time interval, but the standard deviation is quite erratic.

Another test consisted in verifying the capacity of the system to deal with messages of different priorities at the same time, assessing their completation times. This is a typical race condition where tasks with different priorities compete for the same resources. Three computers acted as clients requesting services from the server. Messages were continuously generated in order to increase the possibility of concurrent requests with different priorities within the server. The results of the test are reported in table 10.8. In the case of interpreted Java it is possible to see that the race for resources does not affect high priority messages, it slows down file transfers by about 30 % and doubles the average time needed for a general message to be executed. The large value of the standard deviation indicates variability of the transmission times. In the case of the JIT compiled Java high priority messages take, on average, the same amount of time to return to the low priority ones; and this time is about double that of the reference one. Since the Bytecode used during the test is the same it is likely that the problem of this unexpected behaviour is located inside the Kaffe virtual machine.

In the previous test the data file transmitted was set equal to 100 bytes. In order to assess how the size of the data transfer affects the response time of the server, the experiment was repeated with file transfers of different sizes. The test was performed only with the interpreted Java due to the problems that Kaffe revealed in the previous test. The results are shown in table 10.9. It is possible to notice that the high priority messages do not suffer any appreciable delay related to the size of the data program transferred. Database transmissions are always slowed down by concurrent presence of high priority messages. On the other hand

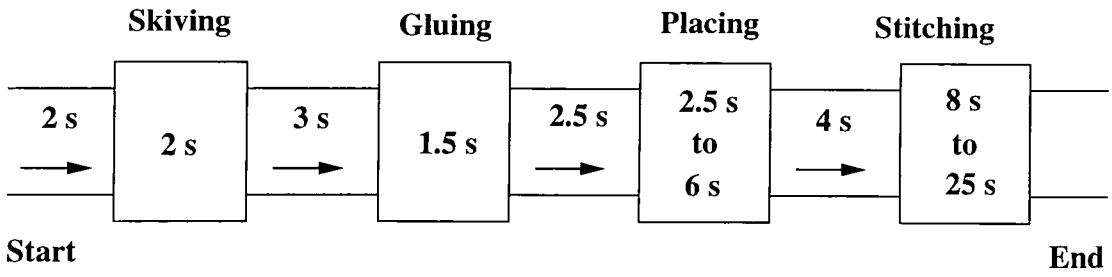


Figure 10.4: Processing times associated with the simulated manufacturing cell.

the delay of the low priority messages first increases, but then decreases. The reason of this phenomenon is likely to be due to the fact that the database transfers become less frequent and therefore they are less likely to interfere with low priority messages. However it is possible to see that the standard deviation associated with the low messages increases. This means that even if they take on average less time to return, the required time has larger fluctuations, indicating possible interferences with concurrent database file transfers.

10.7 Manufacturing simulation

The last experiment involved the simulation of a manufacturing environment controlled by the designed system. It was decided to simulate the cell for the assembly of the first stages of the uppers of the shoes which was described in section 4.6. However the results can be easily extended to the other cell as well. Each of the 80486 machines connected in the local network simulated a manufacturing device while the Pentium acted as manufacturing control system. A set of realistic processing times were used in the simulation (see figure 10.4).

The aim of the simulation was to assess the capability of the system to share information about the component flow, the capacity of the Production Activity Control, of gathering a representation of the status of the cells, and of the possibility of providing remote storage for the data programs.

The simulation starts when the name and the quantity of the components to be processed are inserted in the Production Control System. The schedule is gen-

erated, and then the Dispatcher takes control of sending the scheduled jobs in production. When it is time for a job to enter in production, the Production Management system sends a WRITE message to all the devices informing them of the identity of the component, its arrival time and processing time. Every device inserts this information in the list of the incoming Jobs. At soon as there is an entry in the incoming job lists each device retrieves it and reads the related information. It waits in the IDLE status until the time of the component's arrival. When this time is reached the device changes its status from IDLE to OPERATING and informs the Monitoring system of this fact using a UNSOLICITED_STATUS message. Then it sleeps for a time equal to the processing time of the component. Afterwards it changes its status to IDLE, and informs the Control system of the change of the status and of the time spent on the component using a WRITE message. Then if the device is not the last one in the cell it sends a WRITE message to the following device informing that the component has been successfully processed and it is approaching. Then the next entry of the incoming jobs list is retrieved and the cycle restarts. When a device receives notification that a new component has been placed in production it checks to see if the program data for its processing is already present in memory. If this is not the case the database is queried in order to retrieve this information. During the operations the Polling system included in the Management System is active, periodically queries the status of the connected devices.

During the simulation the times associated with the different messages were recorded. The results were taken running a simulation where 30 pieces belonging to 7 different components were batched in the input tray of the system. Three of the components were single,while the other four had to be processed in pairs to generate the required output. Since no specific program data was incorporated into the database and for files up to 7000 bytes the processing time is almost independent from size of the message, generic queries to the database and data file transfers of 1000 bytes were performed when a file request was received. It was not possible to time the INFORMATION_REPORT message since it does not require acknowledgment.

It was decided to run the simulation using only the interpreted version of Java since the tests in the previous section suggested that Kaffe is not able to handle correctly incoming messages with different priorities.

The average time needed by each of the stations to send a WRITE (OPERATING_TIME_DAT to the the Production Management System were monitored. This messages informs that a piece has successfully completed, together with the time spent to process it. The results follow in table 10.10.

The values measured are comparable on all the devices. The time required to

message type	client	server	mean (s)	stdev	reference mean (s)
WRITE	80486	Pentium	0.145	0.0338	0.106
WRITE	80486	Pentium	0.156	0.0409	0.106
WRITE	80486	Pentium	0.160	0.0432	0.106
WRITE	80486	Pentium	0.147	0.0390	0.106

Table 10.10: Average time to send a WRITE message

perform the operation is about 50% greater than the reference value.

The results of the time required by a device to load a data file are reported below in table 10.11.

These messages have a higher priority in comparison with the others used during

message type	client	server	mean (s)	stdev	reference mean (s)
UPLOAD	80486	Pentium	0.348	0.0940	0.343
UPLOAD	80486	Pentium	0.332	0.0643	0.343
UPLOAD	80486	Pentium	0.368	0.0880	0.343
UPLOAD	80486	Pentium	0.375	0.0573	0.343

Table 10.11: Average time to perform an upload operation of a data file

the simulation, and their times are similar to those taken during the test of the performance of the database with only one client, the reference mean.

The messages WRITE(APPROACHING_COMPONENT), which are sent from one device to the following one for reporting that a component has been successfully processed and it is approaching, showed the performance reported in table 10.12.

An increase in the average time required to execute the service is observed, but

message type	client	server	mean (s)	stdev	reference mean (s)
WRITE	80486	80486	0.165	0.0215	0.143
WRITE	80486	80486	0.190	0.0324	0.143
WRITE	80486	80486	0.172	0.0314	0.143

Table 10.12: Average time to send a WRITE message

it is closer in value to the reference time compared to the other WRITE messages performed by the devices. This is to be expected because these other devices are not subject to the same level of load as the Control Management System.

The time required by the Production Management System to send a message to a station to inform it that a new component has been put in production follows in table 10.13.

It is possible to see in the first line that the performance is quite poor in com-

message type	client	server	mean (s)	stdev	reference mean (s)	random generator
WRITE	Pentium	80486	0.288	0.0952	0.108	absent
WRITE	Pentium	80486	0.193	0.0583	0.108	present

Table 10.13: Average time to send a WRITE message

parison with the reference value. This is because as soon as the system starts informing the devices that a new component has been placed into production, if the device does not have the necessary data program it immediately requests the file from the database. These database requests may produce delays in the other tasks. This fact is the cause of the large value of the mean and of the standard deviation associated with the WRITE instruction. However if the devices which need to retrieve a program from the database wait for a random interval between 0 and 2 seconds before making the query an improvement in the average time of the service is reported. The introduction of this random wait time improved the performance of the database response times of about 10%.

Finally the time required by the poller for servicing a STATUS message is shown

in table 10.14.

This value is smaller than the one required by the WRITE message since this service

message type	client	server	mean (s)	stdev	reference
STATUS	Pentium	80486	0.147	0.0452	0.105

Table 10.14: Average time to send a STATUS message

is less likely to be executed at the same time as a database request and therefore delays are less frequent .

10.8 Conclusions

The testing process carried out in this chapter was aimed to provide an incremental analysis of the performance of the system. First an assessment of the performance of the basic component of the application (the TCP protocol and the database) was performed. Then the times associated with single messages and groups of messages were taken. Finally a complete simulation of a workcell was carried out. This approach allowed a better understanding of the relationships between the components and provided guide for explaining results on the base of the knowledge of the previous behaviour in simpler and more controlled conditions.

The results of the workcell simulation reported here show the expected execution times of the different messages required by the system to perform its activities. The set of messages used is realistic and match the communication requirements proposed by BU. In general it is possible to see that the messages from the computer hosting the Process Control System and the Database suffer from a larger delay. This suggests that these must be implemented on a fast platform, or on two different platforms. However in the second case all the accesses to the database made by the control system would need to be performed using query messages (the definition of a subcode of the WRITE message would serve the scope). Otherwise a distributed database could be employed. The time associated with the messages is in all the

cases in the order of one second. This is believed to be compatible with the system time requirements. However in the case of error messages it has been seen that they are always promptly executed and therefore a response in the order of 0.1 seconds is to be expected.

In future the use of JIT compilers and Java compilers will contribute to boost the performance of Java applications. The tests reported in this chapter confirm the value of using these techniques for improving speed, but they revealed some problems as well. However it is important to remember that the present versions of Kaffe and Toba are still in their *beta* stage and will be surely improved in future.

The implemented manufacturing support system can be modified to incorporate a large range of system models and configurations. Thus it provides a reusable framework which can be used for the design and implementation of support systems for a wide range of manufacturing workcells present on the shopfloor, reducing the development time and costs.

Chapter 11

Conclusions

11.1 Introduction

The rapid flow of information in an industrial environment is essential in order to achieve complete integration and control of the manufacturing processes. Nevertheless at present the majority of devices are still used as stand alone machines, and they do not take advantage of the possibilities offered by a communication link to improve the manufacturing process. The main problem is that the operational control of an automated manufacturing system is a complex activity which requires the interaction and coordination of different subsystems. At present the largest part of the software for achieving automated manufacturing is rewritten for each application. This requires the custom specification, design and development of the overall control support system. This imposes high development costs which are not matched with immediate evident benefits. In addition the development of custom applications introduces the dependency of the automated system on a specific organization or vendors. These factors do not facilitate the spread of integrated control systems in manufacturing environments. This research has recognized these problems and has analyzed the issues connected with the implementation of an integrated control system for manufacturing applications. As a result a simple, modular, inexpensive and portable system was proposed and implemented on a simulated workcell. The area of footwear manufacturing was an

appropriate example to use for the work as recently an increasing number of sophisticated devices, which could take advantage of a network connection, have been developed or introduced on the market

11.2 An integrated control system for automated manufacturing

A integrated control system for manufacturing applications is essentially composed of four elements. The devices, the control system which monitors and coordinates the industrial machines, the database which provides the support information to the whole system, and the communication system which allows the flow of information among the connected parts.

The research carried on during this project has shown that the activities of the control system are dependent on the specific process controlled. Similarly the software needed by a device in order to perform its activities is essentially machine specific. However the analysis of a general manufacturing cell has identified within these components some modules which are application independent. This means that it is possible to build modular software which is able to confine the machine specific segments in specific segments of code. This allowed the definition of the essential modules required in order to implement a manufacturing support system, and highlighted their interrelationships. During all these stages only few implicit assumptions about the final cell and the parts produced were made. The design stage resulted in the definition of flexible software which could be adapted to the particular needs of each connected node. This is useful in the case of cell reconfiguration. Since some software always needs to be replaced every time a new machine is introduced it makes better commercial sense to implement modular systems which allow upgrades with the required functionality [Sar93].

The design of the database system showed that it is necessary to have clear specifications of the system for properly defining the structure and the relationships among the structures storing the relevant information. However the Object

Oriented design approach showed that it is possible to define a database system which allows flexibility and future expansion without the need of major alterations.

Once the devices, the production control system and the database had been successfully modelled, the communication system required for their interconnection had to be designed. Analyzing the problem of defining a communication protocol for industrial environments it was possible to identify a series of stages and guidelines necessary for the development of a successful design. This knowledge has been defined in a framework for rapid protocol development, RPD, which, starting from the definition of the problem, allows the complete development of a communication protocol in a series of incremental steps. The approach allowed a structured design of the communication system reducing the time required and improving the result.

At the early stages of the protocol development, the selection of the communication system had to be taken. The presence of embedded PCs inside the industrial machines suggested the possibility of using a Ethernet network connection. This LAN is the almost ubiquitous way used for connecting PCs together. An extensive survey highlighting the strengths and limitations of Ethernet in industrial environments was performed. This was followed by a series of tests which highlighted the performance of Ethernet used in conjunction with the Internet protocols on a series of different platforms. This suggested Ethernet as a feasible alternative to fieldbusses. The advantages of this network include speed, reliability, diffusion, and ease of use. The tests performed suggested that even if it is not inherently deterministic, it could be used in real-time applications in the case of short, lightly loaded networks. The differences in performance of the protocols and their unexpected behaviour suggest that testing procedures are essential for validating the proper behaviour of the selected protocols. Ethernet proved to have several advantages when compared with the other fieldbusses, and it could provide a valuable alternative to fieldbusses transmission system in all the cases where complex industrial machines embedding PCs are present.

After the selection of the lower layers of the protocol the upper ones were developed. The proposed framework allowed the definition of a simple, flexible and expandable communication protocol, where a limited number of messages, orga-

nized in a hierarchical structure, was defined. The whole process led to the definition of a protocol based on existing technology which could be used to fulfil the requirements of the footwear manufacturing industry. However small alterations and expansions would make it suitable in other areas of manufacturing automation as well.

Starting from the definition of the classes and relationships and within the constraints imposed by the defined transmission protocol, the whole manufacturing support system was developed. The database development did not present any problem and showed the potential of Object Oriented databases in organizing the support information for a manufacturing system. The decision of selecting Java to implement the other modules of the manufacturing support system proved the potential of the language in manufacturing applications. An extended analysis of Java for the implementation of *real world* applications highlighted the advantages and the limitations of the current version of Java and located areas where future developments are necessary. The implementation of the system in this language confirmed the impression that Java could become a promising language for the implementation of manufacturing applications. In particular the platform independence feature of the language allows the immediate porting of applications to systems with different features. The object oriented features and the modularity of Java allow the development of flexible programs. In addition since Java applications are made by modules (`file.class` files) which are loaded when needed, it is possible to modify them without the need of altering the whole application, reducing the efforts required for updating the software. The research showed that the main current limitation of Java is its inability to directly access the hardware. However, recently, Sun Microsystems has announced the release of a new version of Java designed for embedded applications. Although the full specifications are not yet known, it is likely to offer features for addressing the hardware and real-time support.

The test of the system has validated the approach used for the definition of the manufacturing support system and the transmission protocol. The manufacturing cell simulation has shown that the times associated with the manufacturing

support system operation are compatible for its use in real applications where the response times are not too tight. When the application was executed using a JIT Java compiler a considerable improvement of the performance of the system was found. This suggests a future wider use of alternative approaches to interpreted Java in order to improve system performance.

The manufacturing support system were tested with a particular workcell, but its flexible structure allows it to be adapted to control any workcell for the footwear assembly, and to other types of industrial processes as well.

The use of Ethernet and Internet protocols allowed the definition of a simple, cheap and flexible transmission protocol, and the decision to use Java for implementing the system conferred similar features to the manufacturing control system as well.

This thesis has shown that it is possible to implement an efficient, simple, flexible and inexpensive manufacturing control system based on readily available equipment, and taking advantage of the features that an innovative programming language like Java can offer.

This research has developed an original framework for the implementation of an adaptable communication and control system for manufacturing applications bringing together a broad range of disciplines and novel technologies in a new way. During this process several problems have been encountered and original solutions proposed. A novel approach for solving a particular scheduling problem has been developed. Promising preliminary research which showed the possibility of transferring glue using electrostatic deposition has been carried out. Finally a useful technique for the development of transmission protocols, especially suited to manufacturing applications, has been designed.

11.3 Future work

The current version of the system has been implemented and tested on a set of networked computers. The natural development of the project would be to transfer the proposed system to a real manufacturing cell. The "BU Rink" would represent

the ideal target for this test, and real data on the performance of the system could be gathered. It is suggested that during this porting activity the new features offered by the new versions of Java should be included in the software, in order to increase the robustness of the code. For example it would be useful to insert a timeout to the sockets which are waiting for reply messages in order to allow recovery procedures in the case of message loss.

The decision to divide the system in two modules, the Device and the Interface was taken since Java is not yet able to directly address the hardware. However this is likely to change in future with the introduction of the embedded version of Java. If the language offers sufficient support for accessing the hardware the integration of the two modules into a single unit could become feasible, reducing complexity and improving performance.

Appendix A

Performance Measurements

A.1 Introduction

During the research the performance of TCP and UDP implementations of several operating system have been evaluated. Due to the impossibility of including all the results in the main body of this thesis some of the more meaningful graphs have been inserted in this appendix.

The test of the two protocols was performed always using pairs of identical machines, so as to be able to compare the differences in performance of the transmission protocols with different operating systems on platforms with different processors and architecture.

The details of the computers and operating system used follow:

- BSDI BSD/OS 2.0 on Intel 80486DX2 66 MHz with 32 Mbytes RAM, 1 Gbyte SCSI HD, WD8003 Ethernet card;
- BSDI BSD/OS 2.1 on Intel 80486 33 MHz with 8 Mbytes of RAM, 1.2 Gbytes IDE HD, WD8003 Ethernet card;
- HP-UX A.09.05 A9000/710 on a HP Apollo Series 700 (HP 710), with 16 Mbytes RAM, custom Ethernet card;

- Linux 1.2.13 on Pentium 75 MHz with 16 Mbytes of RAM, 1 Gbyte EIDE HD, SMC Ultra Ethernet card;
- Linux 1.2.13 on on Intel 80486 33 MHz with 4 Mbytes of RAM, 200 Mbytes IDE HD, WD8003 Ethernet card;
- Linux 1.2.13 on on Intel 80386 20 MHz with 4 Mbytes of RAM, 41 Mbytes IDE HD, WD8003 Ethernet card;
- Linux 2.0.29 on Intel 80486 33 MHz with 8 Mbytes of RAM, 1.2 Gbytes IDE HD, WD8003 Ethernet card;
- SunOS 4.1.3 on a Sun Microsystems IPC workstation (25 MHz) with 24 Mbytes RAM, custom Ethernet card;
- Solaris 2.4 (SunOS 5.5.1) on a Sun Microsystems IPC workstation (25 MHz) with 24 Mbytes RAM, custom Ethernet card;
- Solaris 2.4 (SunOS 5.5.1) on a Sun Microsystems Sparc Ultra 1/140 (140 MHz), 64 Mbytes RAM, 2 Gbytes HD and custom Ethernet card;

The experiments with the Linux operating system and BSDI BSD/OS 2.1 were performed on an isolated network. All the other test have been executed during the night at times when the overall network usage was very low, on unloaded systems (the only user's processes active on the platform were the ones involved with the test program) in order to minimize the effect of the local traffic. For every message length, 100 trials were performed (in sets of 25 at different times during the night to reduce the effects of temporary traffic for non-isolated networks). It was decided to use the default values associated with the protocols. In order to avoid difficulties of clock synchronization between the different machines used in the experiments, all the measurements were made by taking the "round trip time" (double transit time) of the data from host A to host B and back again.

The timing procedures for both TCP and UDP protocols are very similar, due to the intrinsic resemblance of the two protocols. The software used for taking the measurements consists of two programs; the first one runs on the server machine

and the second one on the client. The software on the server creates a socket and binds it to a pre-defined port number and waits. The client software, for each message sent, creates a socket, binds it to the same port number of the server, and sets up the buffer to accommodate the incoming data. Then the timer starts, and the client sends the message to the server, which then sends it back. When the whole message has been received by the client, the timer is halted and the delay is calculated. Subsequently, the buffers are freed and the sockets closed. The cycle repeats itself again for the next transmission.

A.2 Results

In figure A.1 and A.2 the performance of the UDP and TCP protocol on different platforms running Linux 1.2.13 is reported. It is possible to notice that the actual performance of the protocol is strongly influenced by the speed of the processor. In addition it is possible to notice that the round trips times are almost always repeatable. A full comment on these graphs is presented in [BP96].

In figure A.3 a comparison of the implementation of the TCP protocol of Linux 1.2.13. and 2.0.29 on identical platforms (with the exception of the RAM) is drawn. It is possible to notice how the new version of the operating system achieves greater performance. It is possible to reach this conclusion observing the round trip times for small packets, where the difference in memory between the two computers is not important since all the OS can be maintained in RAM and no swapping is necessary. However it is possible to locate three irregularities associated with the graph produced by Linux 2.0.29. As already mentioned in section 3.4 the problem is associated with buffering strategies since as soon as the Nagle algorithm is disabled the problem disappears. In figure A.4 it is possible to see how the throughput of Linux 1.2.13 on the Pentium is dependent on the actual size of the message, since it increases steadily until 50 Kbytes.

In figure A.7 and A.8 the performance of the Solaris and SunOS operating system on different platforms is shown. It is possible to notice that the UDP graphs on

the Sparc IPC workstations are quite scattered, while on the Sparc Ultra the graph is consistent. A similar behaviour is visible on the TCP graphs. Solaris on Sparc IPC performs better than SunOS for small packets, but it presents a much more scattered behaviour for large packets. This suggests that some of the problems related to the scattering are platform dependent due to kernel activity. The Sparc Ultra is a powerful, resource rich platform in comparison with the Sparc IPC, and kernel activity takes a minor fraction of the whole CPU time.

In figure A.5 and A.6 the performance of two versions of BSD/OS on comparable platforms is reported. It is possible to notice that while there are an abundance of scatter points in the UDP graph associated with BSD/OS 2.0, this problem is not evident in BSD/OS 2.1. However the TCP graph of both the versions of the OS presents the same peculiar behaviour. The irregularity in the BSD/OS 2.0 graph associated with the packets of 200 bytes has been associated with the buffering strategy of the OS since it disappears when the Nagle Algorithm is disabled.

In order to try to locate more precisely the source of the problem of the irregularities of BSD/OS more than 500 trials with packets of 4000 bytes were performed. Timers inserted in the client and server side of the test program provided the readings for assessing the time spent by the system to perform each of the operations. The set up time (which include the time used by the system for creating a new TCP connection, sending and acknowledging the packet with the incoming message size), the send and receive times and the round trip time have been recorded. In addition the traffic on the (isolated) Ethernet had been monitored with the help of an oscilloscope. The traces of the oscilloscope are presented in figure A.9 to A.15, which depict the most common patterns encountered during the experiments. First of all it is possible to notice the variety of graphs recorded, This fact indicates that the process of sending packets is not deterministic and several effects may be involved during the transmission. However it has been noted the set up time is always constant and equal to 24 ms on the client side and 4 ms to the client side. In the server side only part of the process is timed since the main server program starts a new process to service the incoming TCP message, and then returns to listen for new connections. The timers are located in the child process, and therefore it is

not possible to time the first stages of the connection which are executed outside this code. The constant set up time indicated is an expected result since it has been noted that for small packets the round trip time is always deterministic. It is possible to see that the `send()` function always returns before the packets are actually sent along the network. Since the `send()` returns almost immediately it looks as if the time associated with the receive action on the client side is long. However it is possible to notice from the timings and the graphs that the receive function always returns within 5-10 ms after the transit of the packet along the network. This strongly indicates that the send part of the TCP protocol is responsible for the problem. Since it is possible to notice that the distance between the packets in the oscilloscope traces is sometimes repeated it indicates that several effects contribute to the delays of the send function. For example the first part of the traces in graphs A.9 and A.13 are identical, while the second part of graph A.13 is the same as the first part of A.10. The inconsistency of the traces makes extremely difficult to locate the cause of the unexpected performance of TCP, and their repeating patterns suggest that more than one cause may be present.

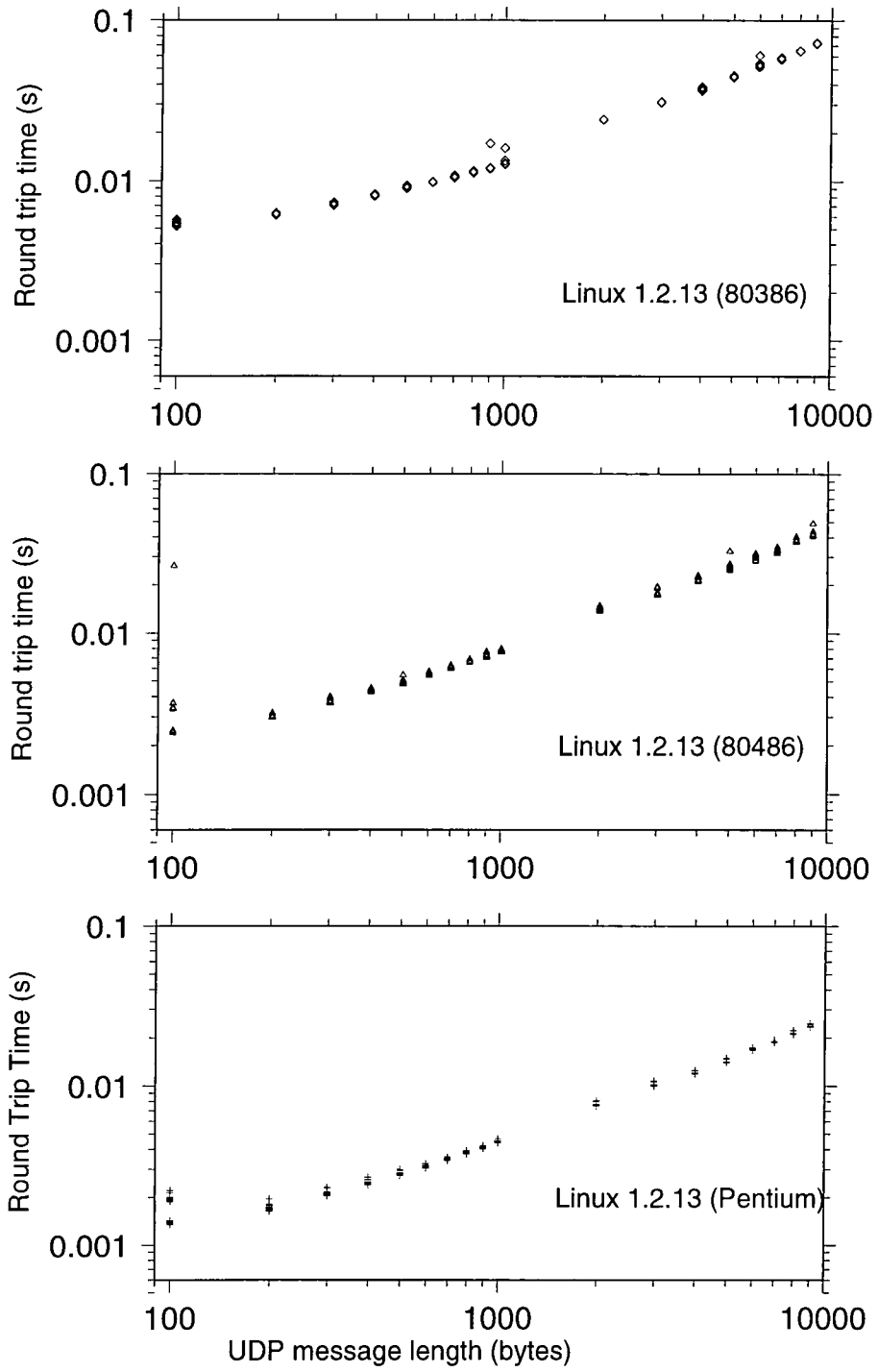


Figure A.1: UDP transmission protocol performance on three different platforms running Linux 1.2.13

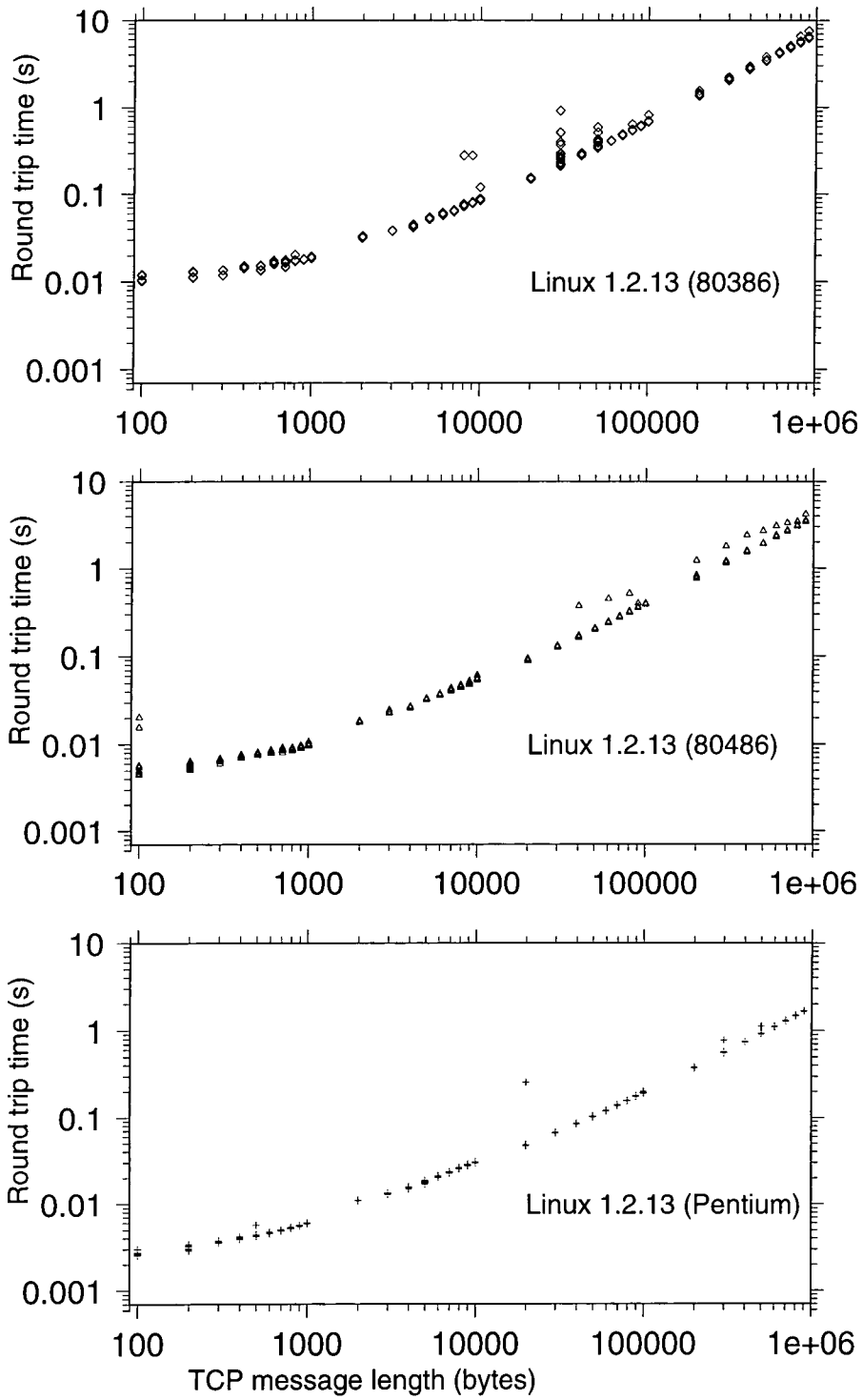


Figure A.2: TCP transmission protocol performance on three different platforms running Linux 1.2.13

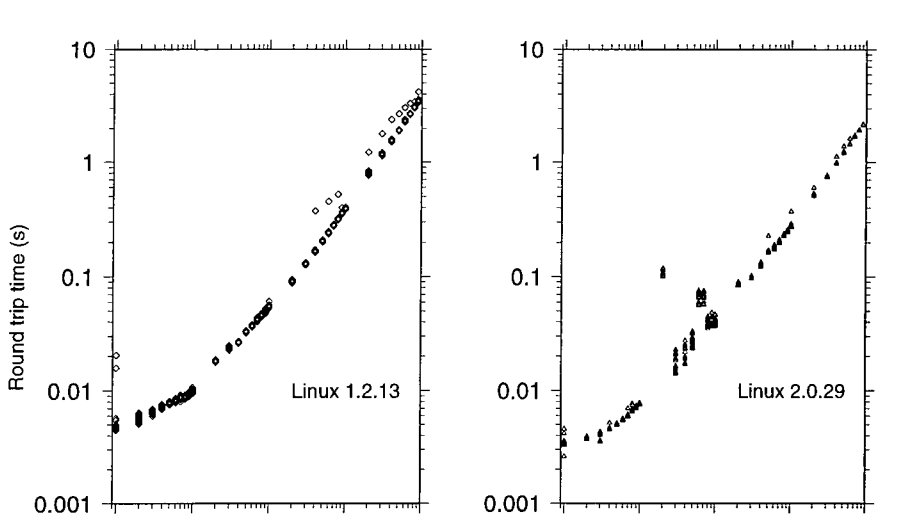


Figure A.3: TCP transmission protocol performance of Linux 1.2.13 and 2.0.29 on Intel 80486

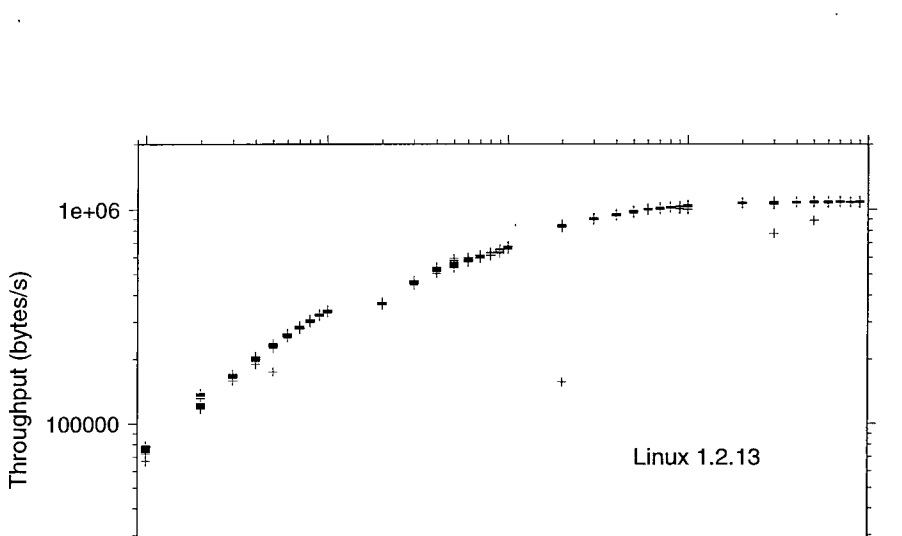


Figure A.4: Throughput of the Linux operating system as a function of the message size

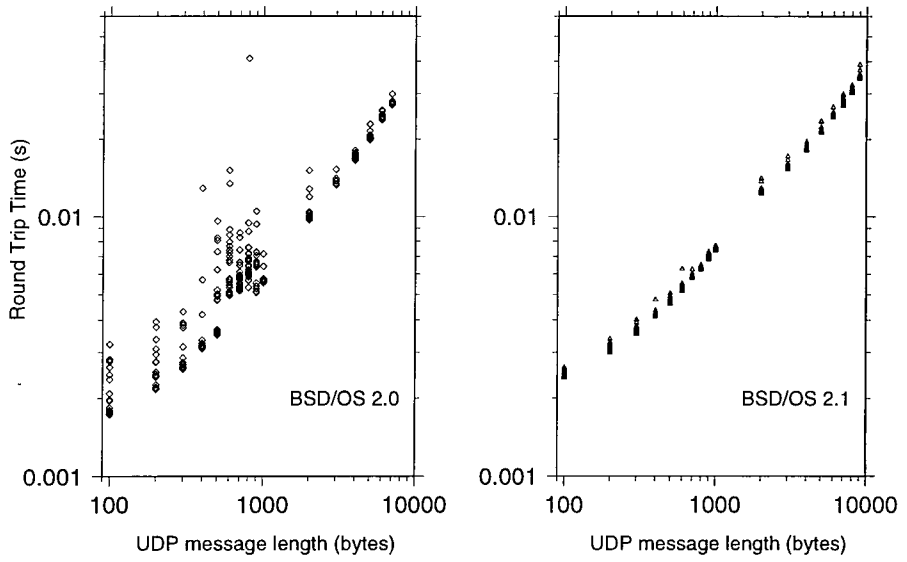


Figure A.5: UDP transmission protocol performance of BSD/OS 2.0 and 2.1 on Intel 80486

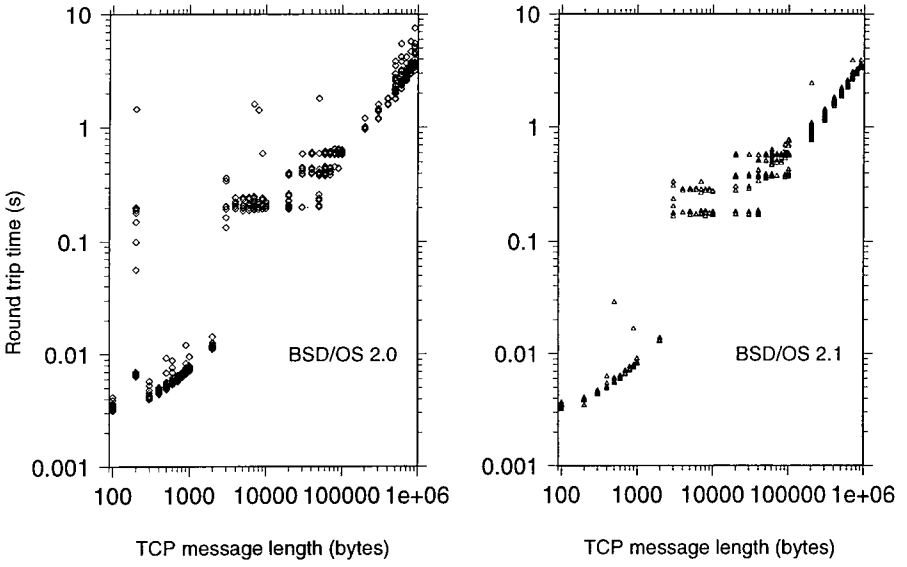


Figure A.6: TCP transmission protocol performance of BSD/OS 2.0 and 2.1 on Intel 80486

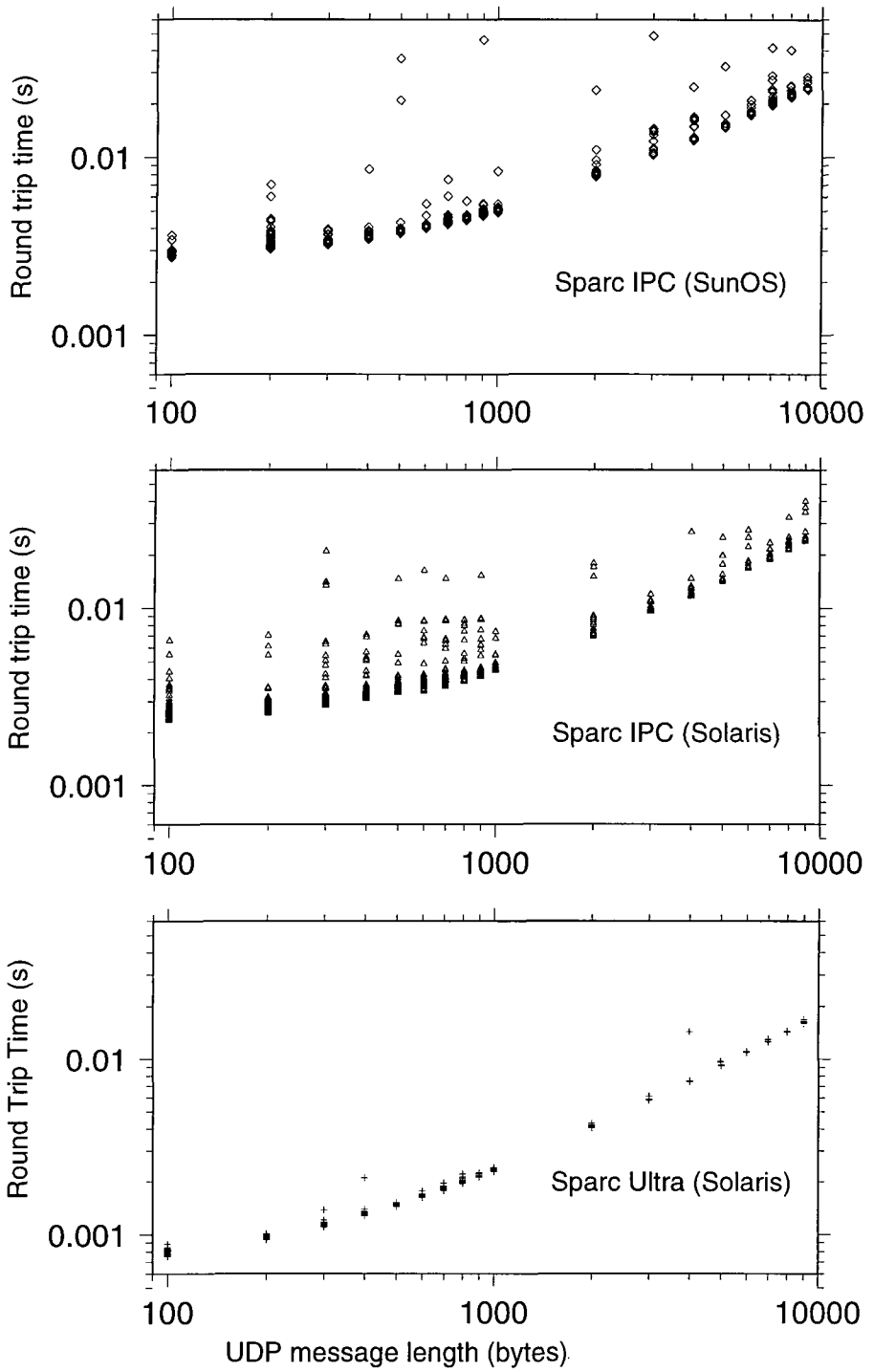


Figure A.7: UDP transmission protocol performance of SunOS and Solaris on different platforms

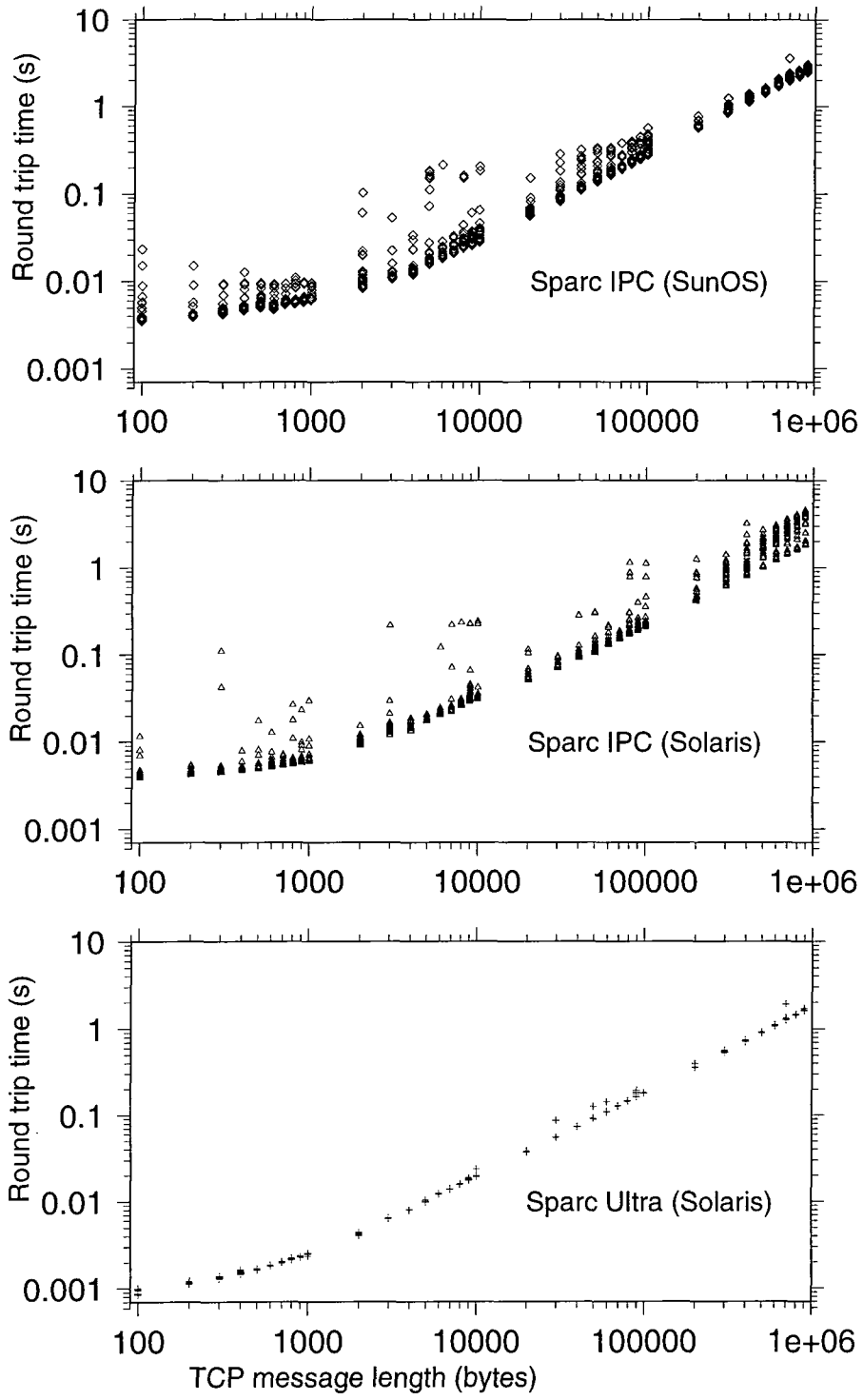


Figure A.8: TCP transmission protocol performance of SunOS and Solaris on different platforms

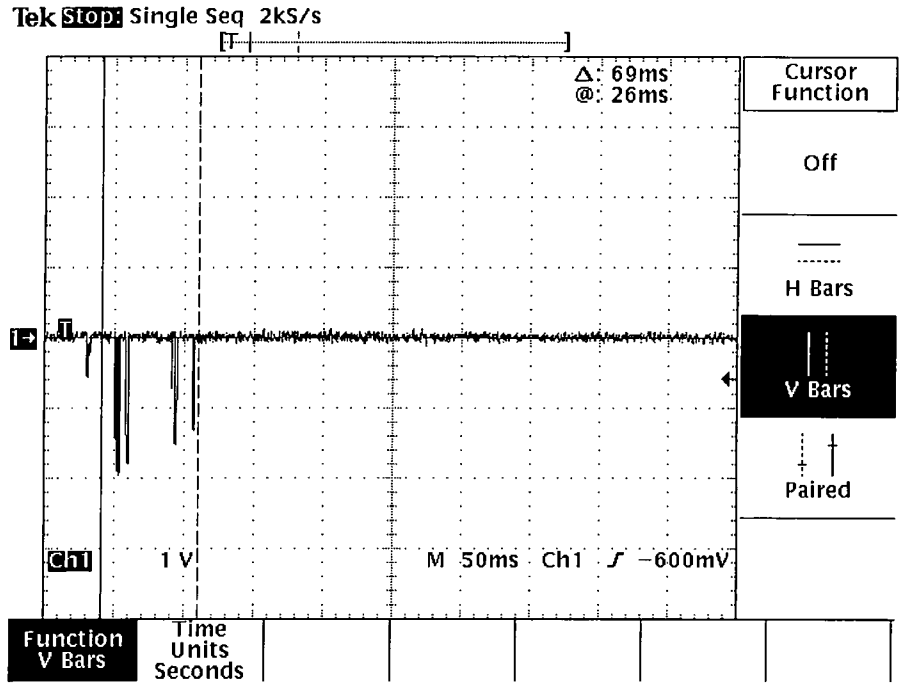


Figure A.9: Client; setup: 24, send: 3, rec: 66 r_trip: 69 – Server; setup: 4, rec: 28, send: 3

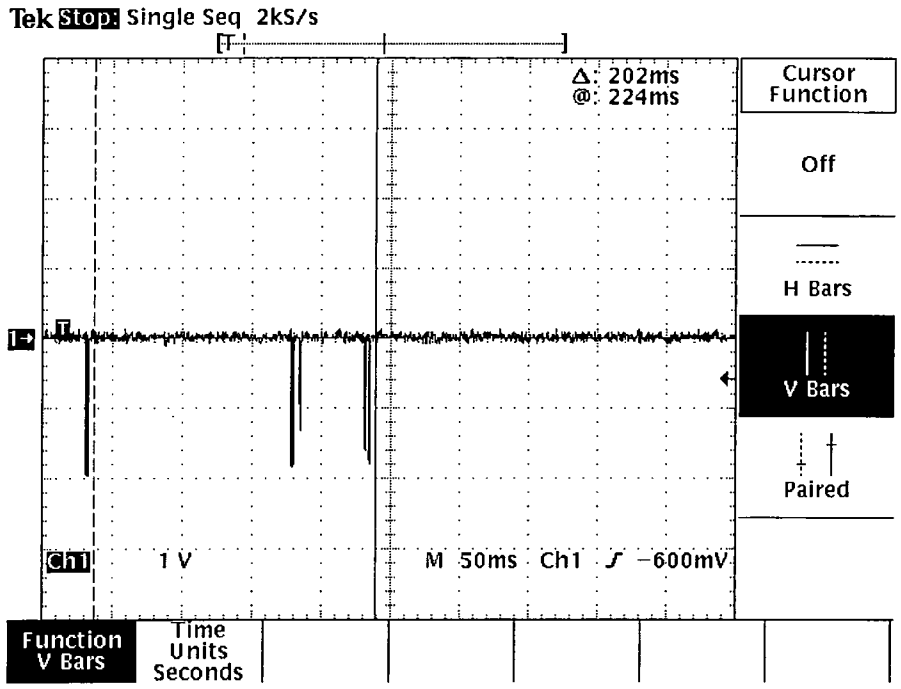


Figure A.10: Client; setup: 23, send: 3, rec: 206 r_trip: 209 – Server; setup: 4, rec: 153, send: 3

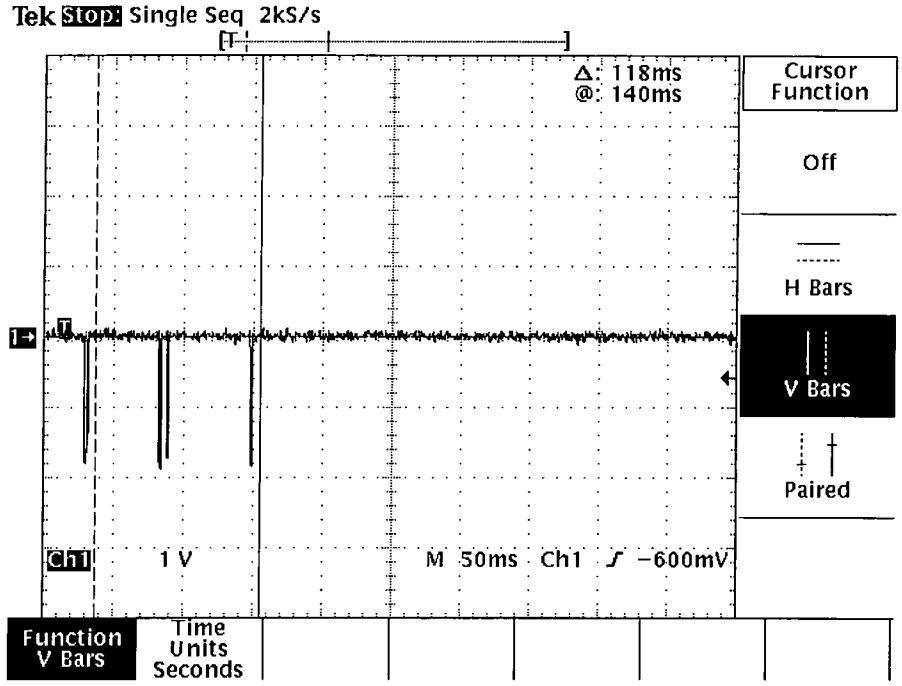


Figure A.11: Client; setup: 23, send: 3, rec: 121 r_trip: 124 – Server; setup: 4, rec: 59, send: 3

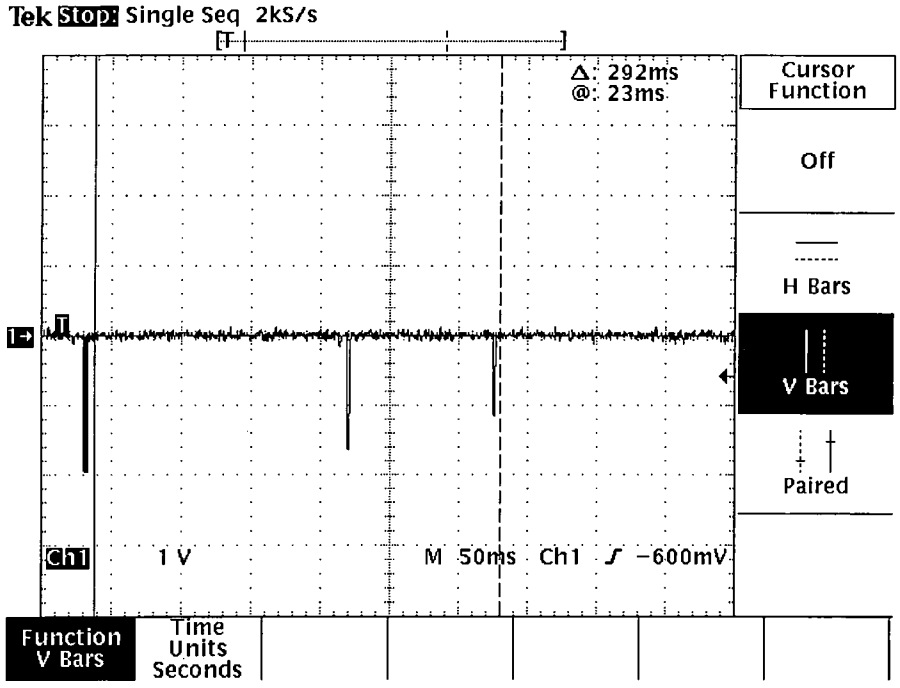


Figure A.12: Client; setup: 24, send: 3, rec: 298 r_trip: 301 – Server; setup: 4, rec: 190, send: 4

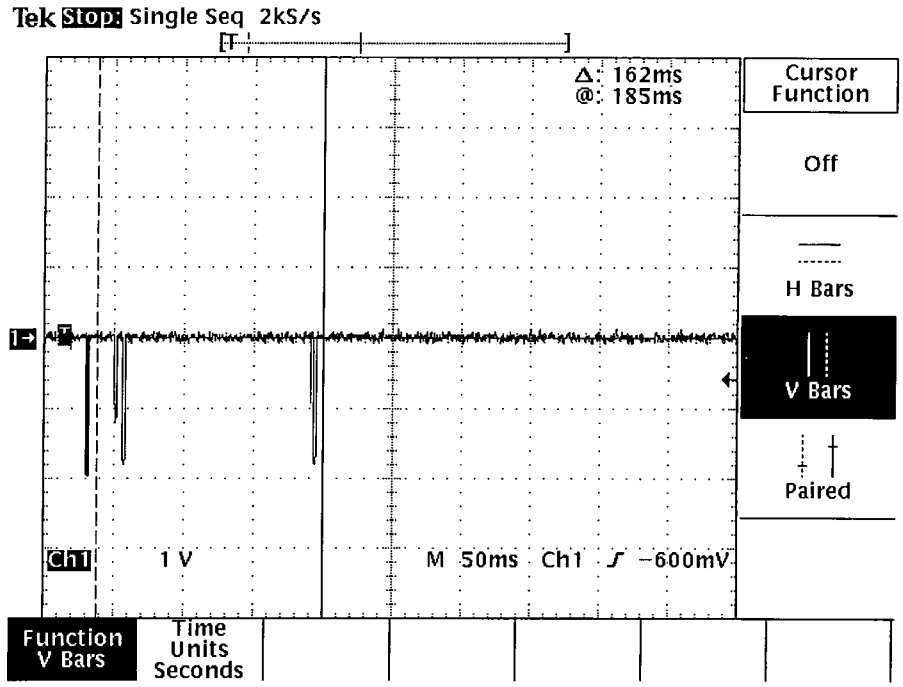


Figure A.13: Client; setup: 24, send: 3, rec: 166 r_trip: 169 – Server; setup: 4, rec: 27, send: 3

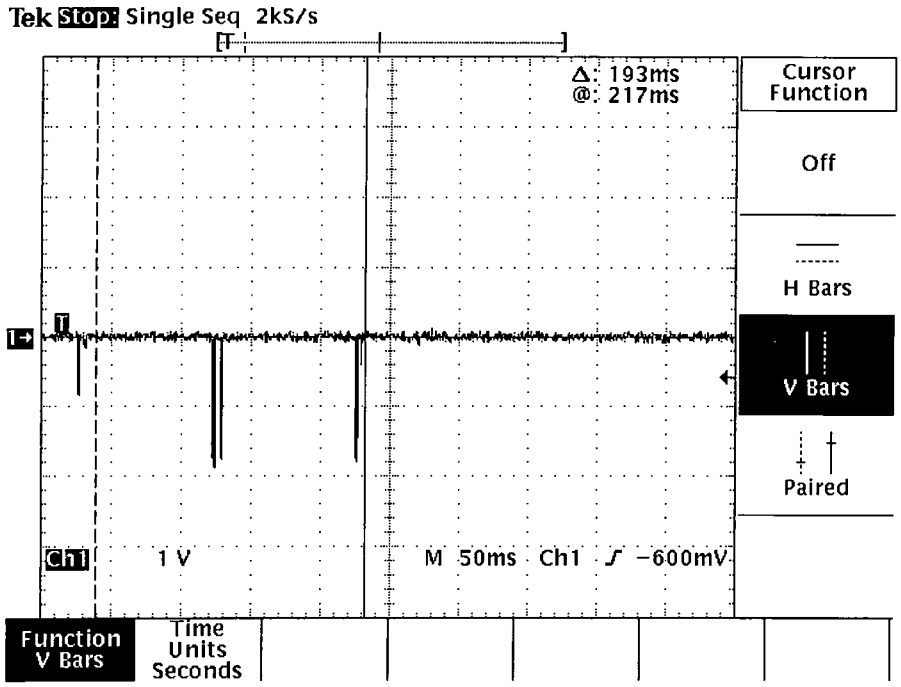


Figure A.14: Client; setup: 24, send: 3, rec: 199 r_trip: 202 – Server; setup: 4, rec: 99, send: 3

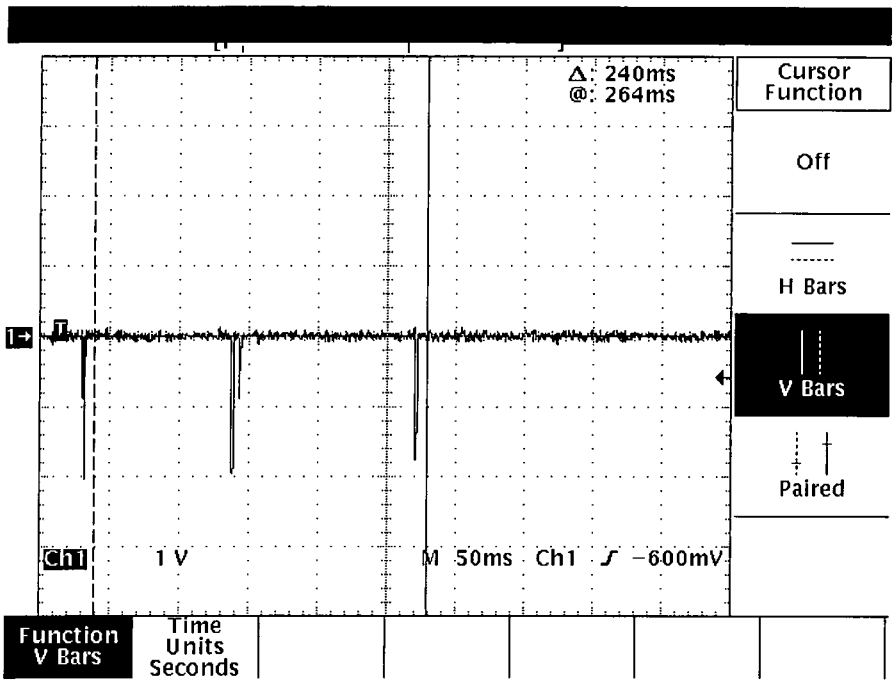


Figure A.15: Client; setup: 24, send: 3, rec: 243 r_trip: 247 – Server; setup: 4, rec: 113, send: 3

Appendix B

Mathematical details of the scheduling algorithm

The following notation has been used:

n	number of jobs to be scheduled
m	number of machines in the flow shop
t_{ij}	processing time for the ith job on the jth machine
$F_{m(ij)}$	total flowtime for job i followed by job j on m machines
$F^*_{m(ij)}$	simplified flowtime for job i followed by job j on m machines

In a two-job, two-machine environment where job i precedes job j, the two possible flowtime sequences are shown in figure B.1. From the part A of the figure, if $t_{j1} > t_{i2}$, it is possible to see that the flowtime for job i is equal to $(t_{i1} + t_{i2})$. Similarly the flowtime for job j is $(t_{i1} + t_{j1} + t_{j2})$. The total flowtime is the sum of the two flowtimes:

$$F_{2(ij)} = 2t_{i1} + t_{i2} + t_{j1} + t_{j2} \tag{B.1}$$

Instead if $t_{j1} < t_{i2}$, as in the part B of the figure, the total flowtime is equal to:

$$F_{2(ij)} = 2t_{i1} + 2t_{i2} + t_{j2} \tag{B.2}$$

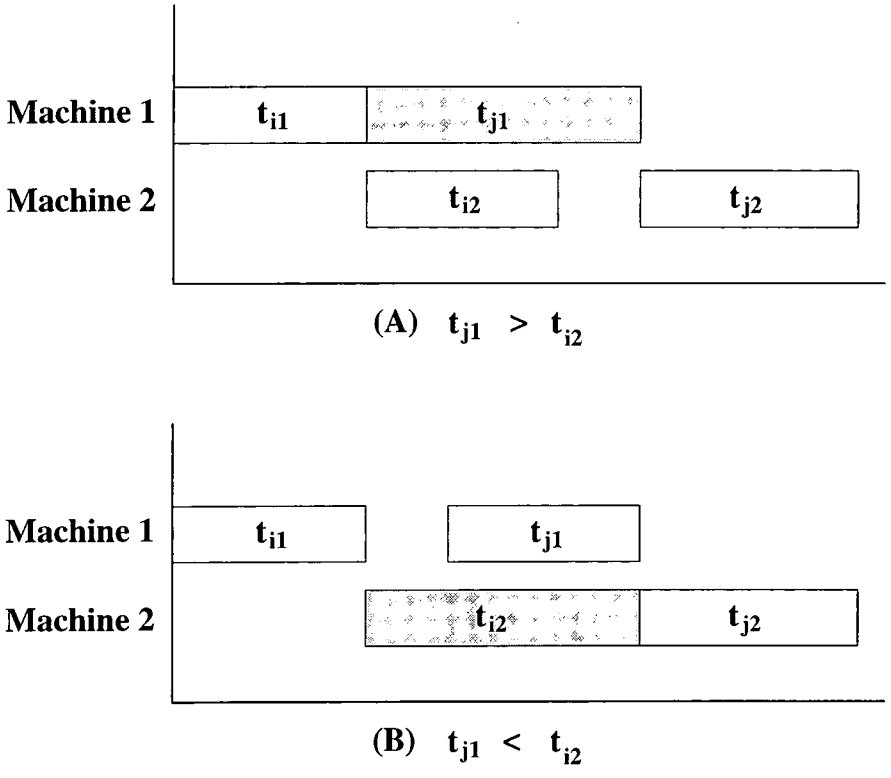


Figure B.1: Gantt charts for the two machine problem

Both the equations B.1 and B.2 are feasible. Therefore the *minimum* total flowtime can be expressed with the equation:

$$F_{2(ij)} = 2t_{i1} + t_{i2} + t_{j2} + max(t_{j1}, t_{i2}) \tag{B.3}$$

Inverting the order of the jobs such as job j precedes job i equation B.3 becomes:

$$F_{2(ji)} = 2t_{j1} + t_{j2} + t_{i2} + max(t_{i1}, t_{j2}) \tag{B.4}$$

It is possible to eliminate the common terms from equations B.3 and B.4, in order to be able to obtain simpler expressions for the following analysis:

$$F^*_{2(ij)} = 2t_{i1} + max(t_{j1}, t_{i2}) \tag{B.5}$$

$$F^*_{2(ji)} = 2t_{j1} + max(t_{i1}, t_{j2}) \tag{B.6}$$

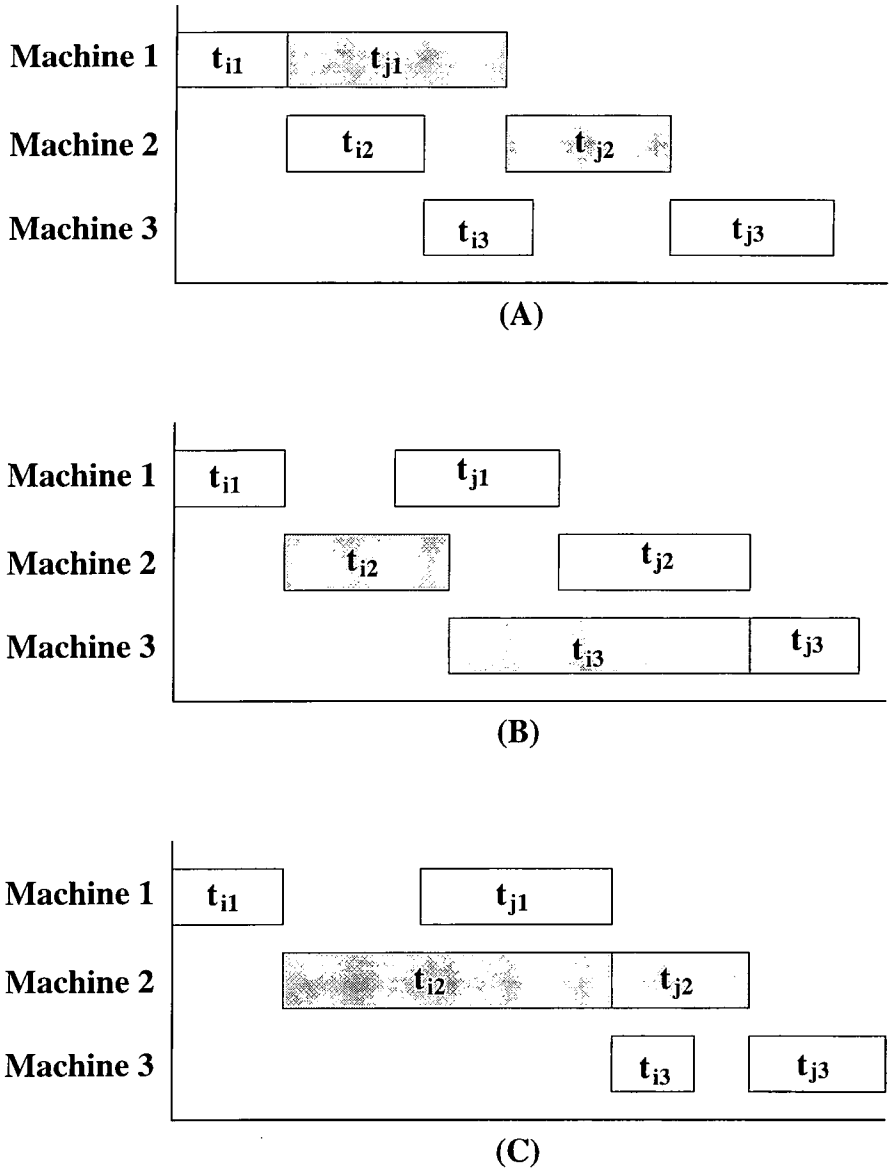


Figure B.2: Gantt charts for the three machine problem

The idea behind the proposed heuristic algorithm suggests the use of equations B.5 and B.6 to find the possible simplified flowtimes, in order to determine the sequence that minimizes the total flowtime. When the schedule is produced the sequence ij or ji is used according to which has associated the smaller simplified flowtime.

The possible cases of an environment where 2 jobs are processed by a set of 3 machines are shown in figure B.2. From the picture it is possible to see that the minimum flowtime is a function of the maximum among the following terms:

$$(t_{j1} + t_{j2}) \quad \text{or} \quad (t_{i2} + t_{i3}) \quad \text{or} \quad (t_{i2} + t_{j2})$$

which can be expressed as:

$$max(t_{j2} + max(t_{j1}, t_{i2}), t_{i2} + t_{i3})$$

The expression of the total flowtime is:

$$F_{3(ij)} = 2t_{i1} + (t_{i2} + t_{i3}) + t_{j3} + max(t_{j2} + max(t_{j1}, t_{i2}), t_{i2} + t_{i3}) \tag{B.7}$$

Similarly to the case of 2 jobs and 2 machines, $F_{3(ji)}$ is the same of equation B.7 with all the i and j subscripts reversed. Removing the common terms form $F_{3(ij)}$ and $F_{3(ji)}$ it is possible to obtain the following expression for the simplified flowtimes:

$$F^*_{3(ij)} = 2t_{i1} + t_{i2} + max(t_{j2} + max(t_{j1}, t_{i2}), t_{i2} + t_{i3})$$

$$F^*_{3(ji)} = 2t_{j1} + t_{j2} + max(t_{i2} + max(t_{i1}, t_{j2}), t_{j2} + t_{j3})$$

It is possible to provide similar expressions for pairs of jobs processes on m machines. In this case the flowtime equation is:

$$F_{m(ij)} = 2t_{i1} + \sum_{k=2}^{m} t_{ik} + R_m \tag{B.8}$$

And the simplified flowtime is:

$$F^*_{m(ij)} = 2t_{i1} + \sum_{k=2}^{m-1} t_{ik} + R^*_m \tag{B.9}$$

where

$$R_1 = 0$$

$$R_2 = t_{j2} + max(t_{j1}, t_{i2})$$

$$R_3 = t_{j3} + max(R_2, t_{i2} + t_{i3})$$

$$R_4 = t_{j4} + max(R_3, t_{i2} + t_{i3} + t_{i4})$$

$$R_m = t_{jm} + max\left(R_{m-1}, \sum_{k=2}^{m} t_{ik}\right) \tag{B.10}$$

$$R_m^* = R_m - t_{jm} \qquad m \geq 2 \tag{B.11}$$

Appendix C

Messages defined in MMS, Profibus and the proposed communication protocol

The following table presents a lists of the messages provided by MMS, Profibus and the proposed communication protocol. The services are grouped in functional categories.

message class	MMS	Profibus	proposed system
Environment and general management	Initiate Conclude Cancel Abort Reject	Initiate Abort Reject	already implemnted in TCP/IP
VMD support	Status UnsolicitedStatus GetNameLis Identify Rename GetCapabList	Status UnsolicitedStatus Identify	Status UnsolicitedStatus Identify
Variable access	Read Write InformationReport DefineNamedVar DefineScatteredAccess GetScatAccessAttrib DeleteVarAccess DefineNamedVarList DeleteNamedVarList GetNamedVarListAttrib DefineNamedType DeleteVariableAccess GetNamedTypeAttrib	Read Write ReadWithType WriteWithType PhysRead PhysWrite InformationReport InformationReportWithType DefineNamedVarList	Read Write InformationReport

message class	MMS	Profibus	proposed system
Operator communications	Input Ouput		
Semaphore management	TakeControl RelinquishControl DefineSemaphore DeleteSemaphore ReportSemStatus ReportPoolSemStatus ReportSemEntryStatus		
Domain management	Download DownloadSegment TerminateDownloadSegment Upload UploadSegment TerminateUploadSegment RequestDomDownload RequestDomUpload LoadDomContent StoreDomContent DeleteDomain GetDomainAttrib	InitiateDownloadSequence DownloadSegment TerminateDownloadSegment InitiateUplaodSequence UploadSegment TerminateUploadSegment RequestDomainDownload RequestDomainUpload	InitiateDownloadFile DownloadFile InitiateUploadFile UploadFile

message class	MMS	Profibus	proposed system
Program invocation management	CreatePI	CreatePI	
	DeletePI	DeletePI	
	Start	Start	Start
	Stop	Stop	Stop
	Resume	Resume	Resume
	Reset	Reset	Reset
	Kill	Kill	Kill
	GetPIAttrib		
Event management	DefineEC		
	DeleteEC		
	GetECAttrib		
	ReportECStatus		
	DefineEA		
	DeleteEA		
	GetEAAttrib		
	ReportEAStatus		
	DefineEE		
	DeleteEE		
	GetEEAttrib		
	ReportEEStatus		
	AlterEE		
	AlterECMonitoring	AlterECMonitoring	
	EventNotification	EventNotification	EventNotification
		EventNotificationWithType	
	AckEventNotification	AckEventNotification	AckEventNotification
	GetAlarmSummary		
	GetAlarmEnrolSummary		
	TriggerEvent		

message class	MMS	Profibus	proposed system
Journal management	ReadJournal WriteJournal InitializeJournal ReportJournStatus CreateJournal DeleteJournal		
Object dictionary management		GetOD InitiatePutOD PutOD TerminatePutOD	

Appendix D

Java performance

D.1 Introduction

Java has been introduced as a platform independent language. When a Java program is processed by the Java compiler, it is transformed into Bytecode rather than native machine code. This means that in order to execute an application written in Java it is necessary to use a Java Virtual Machine, which interprets and executes the Bytecode on the local platform. Therefore the Java Bytecode represents a platform independent format which can be executed on any system implementing a JavaVM. However, recently, alternative solutions have been presented since Just in Time (JIT) compilers and Java compilers have been introduced. JIT compilers transform the Java Bytecode into machine dependent instructions at execution time. This maintains the platform independence feature of the Java Bytecode and increases its execution speed. Java compilers transform Java programs into machine dependent code, increasing performance, but losing platform independence.

Due to the fact that Java is usually used as an interpreted language, Java applications cannot be as fast as the ones generated using a compiled language like C. It has been reported that the language is on the average about 20 times slower than C [Fla96]. Since no data regarding the performance of Java on a Linux platform was available it was decided to carry out a series of tests in order to assess its actual performance. In addition the performance of Kaffe [Pro97], a JIT compiler

for Java, and Toba [PGTNW97], a Java compiler, were tested. Kaffe is a virtual machine which performs JIT code conversion from the abstract code to the host machine's native code. Toba translates Java class files into C source code. This allows the construction of directly executable programs that avoid the overhead of interpretation.

The Java Test Suite (Version 1.0) developed by the Open Group Research Institute, Grenoble, France, was selected for carrying out the performance evaluation. The test suite was developed with the objective of having a set of tools for verifying the correct behaviour of new Java ports. In addition it provides a benchmark for comparing the performance of new ports with the original code from Sun Microsystems. Furthermore it allows the comparison of the performance of Java with C or C++ since some of the benchmarks are available in all three languages. The suite contains five "chapters" named JavaPerf for performance, JavaCompare for comparison with other languages, JavaValid for validation of the ports, JavaSafe for exercising safety features, and Interactive for performing non automatic tests. The test suite can be run in two modes, a batch mode where all the specified tests write their results in HTML pages and an interactive graphic mode where each test can be executed independently and display its result in a window.

A porting of the Test Suite to the Linux OS had to be performed in order to compile it correctly. Moreover some of the tests had to be adapted in order to allow their execution using the JIT compiler and the Java compiler.

All the tests were performed on a Pentium 75 MHz with 16 Mbytes of RAM, 1 Gbyte EIDE HD, running Linux 2.0.25 (Slackeware 3.1 distribution). Since the main interest of the exercise was the comparison of Java performance with compiled languages only the tests included in the JavaCompare section were executed. The `java` interpreter from the JDK 1.0.2 port for Linux, the `kaffe` JIT compiler version 0.84, the `toba` Java compiler version 1.0.b6, and the `gcc` and `g++` C and C++ compilers version 2.7.2 were used.

A brief description of the meaning of the tests and their results extracted from the test suite on-line manual is reported in the next section.

D.2 Comparison of Java, C, C++ program performances

The benchmarks have been subdivided into five subsections according to the language tested feature. The subsections are: Plum tests (arithmetics tests), Byte tests (taken from the Byte magazine tests), memory tests, Input/Output tests and CPU tests.

D.2.1 Plum tests

They consist of arithmetics tests. These benchmarks were placed in the public domain by Thomas Plum from Plum Hall Inc. They are simple benchmarks with several appealing properties:

- They are short enough to type while browsing at trade shows;
- They are protected against overly-aggressive compiler optimizations;
- They reflect empirically-observed operator frequencies in C programs;
- They give a C programmer information directly relevant to programming.

Some minor modifications have been introduced in order to adapt the tests to Java. The benchmarks are:

Operations on integers: This benchmark performs n iterations (default is 1000) of 1000 integer multiplications. It gives the time needed in milli-seconds to execute this loop. The results are reported in table D.1.

Operations on register: This benchmark performs n iterations (default is 1000) of 1000 register integer operations (addition, shift, etc.). It gives the time needed in milliseconds to execute this loop. The results are reported in table D.2.

	Iterations	time (ms)
java	1000	856
kaffe	1000	139
toba	1000	168
c	1000	130
C++	1000	140

Table D.1: Operations on integers

	Iterations	time (ms)
java	1000	836
kaffe	1000	67
toba	1000	100
c	1000	50
C++	1000	60

Table D.2: Operations on register

Operations on long: This benchmark performs n iterations (default is 1000) of 1000 long register operations (addition, shift, comparison, etc.). It gives the time needed in milliseconds to execute this loop. The results are reported in table D.3.

	Iterations	time (ms)
java	1000	881
kaffe	1000	140
toba	1000	199
c	1000	60
C++	1000	60

Table D.3: Operation on long

Operations on shorts: This benchmark performs n iterations (default is 1000) of 1000 short register operations (addition, shift, etc.). It gives the time needed in milliseconds to execute this loop. The results are shown in table D.4.

	Iterations	time (ms)
java	1000	1058
kaffe	1000	103
toba	1000	126
c	1000	70
C++	1000	60

Table D.4: Operation on shorts

Operations on doubles: This benchmark performs n iterations (default is 1000) of 1000 double operations (addition, multiplication, division, etc) . It gives the time needed in milliseconds to execute this loop. The results follow in table D.5.

	Iterations	time (ms)
java	1000	1075
kaffe	1000	684
toba	1000	256
c	1000	160
C++	1000	160

Table D.5: Operations on doubles

Function calls: This benchmark performs n iterations (default is 1000) of 1000 function calls. It gives the time needed in milliseconds to execute this loop. The results of the test are reported in table D.6.

D.2.2 Byte magazine tests

These test are taken from the Byte magazine benchmark collection.

	Iterations	time (ms)
java	1000	2225
kaffe	1000	673
toba	1000	203
c	1000	190
C++	1000	200

Table D.6: Function calls

The Hanoi tower problem: This benchmark computes the time needed to solve tower of Hanoi problem with n discs (default is 20). The number of disc permutations needed to solve the problem is $2^n - 1$. For 20 it means 1048576 permutations. The results follow in table D.7.

	num disks	time (ms)
java	20	9429
kaffe	20	1415
toba	20	887
c	20	550
C++	20	550

Table D.7: Hanoi tower problem

Recursion and memory management: game resolution: This benchmark simulates a "puissance4" game resolution. It will, for each player, compute the best movement with a recursion level given as a parameter (default is 2). Each recursion needs memory in order to copy the game board. The game board size can be increased using the fill parameter (default size 7x7). The results are shown in table D.8.

Drhystone: This test contains many procedure calls which are meant to represent system programming environments. This is a modified version of the dhrystone program. The results obtained with this version are not applicable to other ver-

	depth	size	time (ms)
java	2	7x7	16108
kaffe	2	7x7	6785
toba	2	7x7	3341
c	2	7x7	420
C++	2	7x7	530

Table D.8: Recursion and memory management

sions. The results are reported in table D.9.

	iterations	time (ms)
java	1000	433
kaffe	1000	63
toba	1000	37
c	50000	779
C++	50000	843

Table D.9: Drhystone

D.3 Memory allocation

This test performs allocation and deallocation (in C, C++) of a 1024 bytes array in a loop performed 1000 times. In Java this test gives an indication of the cost of garbage collection since the arrays are allocated and deallocated at each loop. The results are shown in table D.10.

D.4 Input/Output

The test included in this section performs evaluations of the time required by input and output operations.

	iterations	time (ms)
java	1000	113
kaffe	1000	121
toba	1000	371
c	100000	170
C++	100000	240

Table D.10: Memory allocation

PerfReadWrite: This benchmark tests file Read/Write performance. Each write may be followed by a flush. The read/write block size and the number of characters to read/write are program parameters. By default: no flush are performed, read/write size is 1024 bytes, number_of_read/write is 1000, read/write path is "./tmp". The results follow in table D.11.

	iterations	blocs size	flush	Write time (ms)	Read time(ms)
java	1000	1024	false	565	248
c	1000	1024	false	130	210
C++	1000	1024	false	120	210

Table D.11: Input/Output

D.5 CPU

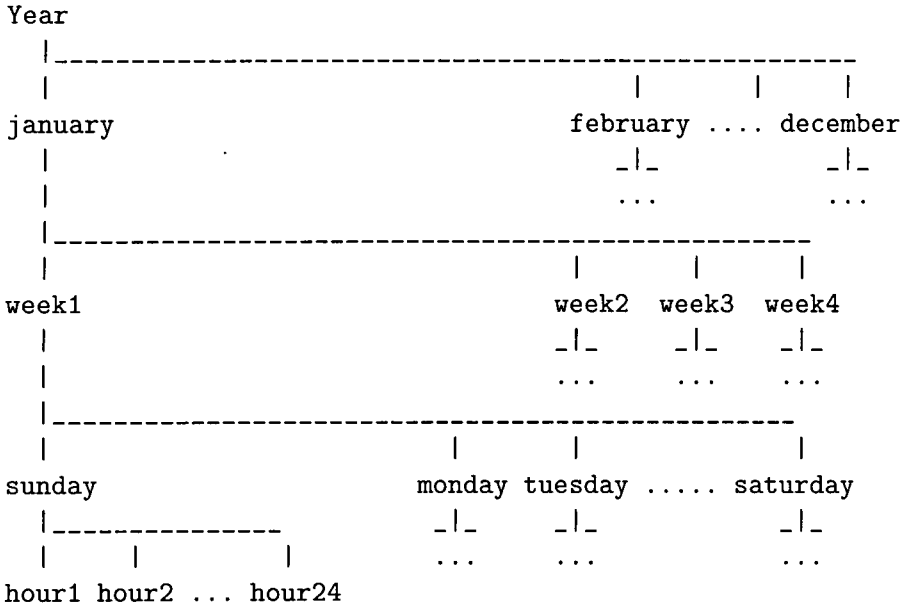
The benchmarks in this section perform tests on language features and assess their CPU time requirements.

Overloading: Test CPU performance for overload and override of methods in classes (C++ and Java). This test has a class with 50 methods with the same name and has a hierarchy of 50 classes with methods overrided. The results are shown in table D.12.

	number loops	override depth	override time (ms)	overload time (ms)	override check
java	1000	50	273	253	true
kaffe	1000	50	187	69	true
toba	1000	50	276	45	true
C++	1000	50	85	27	true

Table D.12: Overloading

List of lists of objects: This test builds a tree describing a Year:



This builds a tree with 8461 small objects. Traversing the tree recursively yields a list of objects contained in the classes year, month, week, day. The results follow in table D.13.

Member access: The aim of that test is to evaluate the access time to members of classes in a tree of classes. the data environment is a tree. The test accesses an integer through members of members of an object:

	iterations	creation time (ms)	listing time (ms)
java	10	855	6240
c	100	31	2349

Table D.13: List of lists of objects

```
class1::member1          class1::member2   ....
     |                        |
class2::member1          class2::member1  .....
     |                        |
    ....                     ....
     |                        |
class9::IntegerData     class9::integerData ...
```

In Java, the first access to a integerData is longer than the other ones (it must calculate the offset) and the others access are faster. In C++, this time is about the same. NOTE: the tree creation may last about 30 seconds because `java` increases its memory space while building the tree. This number is reduced to reduce it to 1 or 2 seconds by specifying `java` that it must initialize its memory space to N bytes (default 2 Mbytes). The results are shown in table D.14

	iterations	creation time (ms)	listing time (ms)	average (access/ms)
java	10	1923	136	37
C++	1000	263	248	2064

Table D.14: Member access

Loop time: This test evaluates time for a loop to do: nothing, an assignation (`a=a`), an `if` with an assignation with a condition (always verified), an `if.. else` with an assignation in both cases. The results follow in table D.15

	number loops	Results (in microsecs/loops) empty loop	assignation	if	if..else
java	1000000	2.513	3.173	5.24	5.652
kaffe	1000000	0.143	0.148	0.256	0.254
toba	1000000	0.214	0.269	0.427	0.428
c	1000000	0.083	0.082	0.124	0.196
C++	1000000	0.082	0.083	0.140	0.139

Table D.15: Loop time

D.6 Analysis

From the data presented it is possible to see that Java performs worse than C and C++ when tested against the benchmarks. The performance of `java` in the arithmetic tests is 5-20 times slower than the one of C and C++. In particular the greatest difference is found when register operations (such as shifts) are involved. However `kaffe` usually has a performance comparable with C and C++ and surprisingly `toba` performs worse than `kaffe` in several tests. This is due to the overhead introduced by the conversion of the Java code into C code. Nevertheless it is possible to see that in the case of operations on doubles and function access `toba` is superior to `kaffe`. When the complexity of the benchmark increases, as happens with the Byte benchmarks, it is possible to see that the `java` performance is 20-40 times slower than C and C++. `kaffe` performs from 3 to 10 times better and `toba` is twice as fast as `kaffe`. The improvement in speed is great, but the execution times are always slower in comparison with C native code. Automatic memory management is inefficient in comparison with direct memory management offered by C and C++. The figures associated to C and C++ are 100 times smaller that the one associated with Java. `kaffe` has a performance similar to `java`, and `toba` is three time slower. This fact explains the slow performance of `kaffe` and `toba` in the recursion and memory management test when compared with C and C++. The Input and Output operations require in `java` a time which is comparable with the time employed by C and C++. The CPU benchmarks show again that Java

is always less efficient than compiled languages. C++ code is 50 times faster than Java in accessing a member of a class. The override time of `toba` is similar to that shown for `java`, while the overload time is similar to that of the C++ code. `java` gives loop times which are more than 20 times slower than C and C++. `toba` is 3 times slower while `kaffe` is about only 2 times slower than C, and C++.

From the results shown here it is clear that JIT compilers and Java compilers sensibly improve the performance over the interpreted version of Java applications. Only in a few benchmarks do they have a performance comparable with equivalent benchmarks written in C and C++. However, the time to create an object does not improve at all and therefore reducing the number of objects created by an application is a good practice in order to improve performance.

Appendix E

Glossary

AGV: Autonomous Guided Vehicle;

ARP: Address Resolution Protocol;

BU, BUSM: British United Ltd., British United Shoe Machinery Co.;

CAD: Computer Aided Design;

CSI: Closed System Interconnection;

CSMA/CA: Carrier Sense Multiple Access/Collision Avoidance;

CSMA/CD: Carrier Sense Multiple Access/Collision Detection;

CSMA/CR: Carrier Sense Multiple Access/Collision Resolution;

DBMS: Data Base Management System;

DSP: Digital Signal Processor;

FIFO: First In First Out;

HD: Hard Disk;

IP: Internet Protocol;

I/O: Input/Output;

JIT: Just In Time;

JavaVM, JVM: Java Virtual Machine;

LAN: Local Area Network;

MAC: Medium Access Control;

MAP: Manufacturing Automation Protocol;

MMS: Manufacturing Message Specification;

NC: Numerically Controlled (machine);

NFS: Network File System;

OMT: Object Modeling Technique;

OS: Operating System;

OSI, ISO/OSI: International Standard Organization Open System Interconnection;

PC: Personal Computer;

PLC: Programmable Logic Controller;

RFC: Request For Comments;

RPD: Rapid Protocol Development;

RTOS Real-Time Operating System;

TCP: Transmission Control Protocol;

UDP: User Datagram Protocol;

VFD: Virtual Field Device;

VM: Virtual Machine;

VMD: Virtual Manufacturing Device;

WWW: Word Wide Web;

Bibliography

[ABM87] S. Aggarwal, D. Barbará, and K. Z. Meth. SPANNER: A tool for the specification, analysis, and evaluation of protocols. *IEEE Transaction on Software Engineering*, 13(12):1218–1237, 1987.

[Ada88] W. M. Adams. Manufacturing network performance and environmental measurements. In *ENE'88*, Baltimore, Ma, June 1988.

[Ada90] W. M. Adams. Why 802.3 Ethernet LANs flourish in manufacturing operations. *Computer-Integrated Manufacturing Systems*, 3(1):53–56, 1990.

[AG96] K. Arnold and J. Gosling. *The Java Programming Language.* Addison Wesley, 1996.

[Alh96] K. Alhola. Katix mini-ip. available at: http://www.funet.fi/~kate/mini-ip.html, 1996.

[Ame92] American National Standards Institute, New York, NY. *Database Language SQL92*, 1992.

[AP82] I. Adiri and D. Pohoryles. Flowshop no-idle on no-wait scheduling to minimize the sun of completion times. *Naval Res. Logistics Quarterly*, 29(3):495–504, 1982.

[Ash88] C. M. Ashworth. Structured System Analysis and Design Method (SSADM). *Information and Software Technology*, 30(3):153–163, 1988.

[BBB+91] A. Bauer, R. Bowden, J. Browne, J. Duggan, and G. Lyons. *Shop Floor Control System - From Design to Implementation.* Chapman & Hall, 1991.

[BBBB91] B. Biron, G. Bel, J. Cavaile O. Baraust, and J. Burrieres. Integrating simulation for workshop control. In *Computer Applications in Production and Engineering.* North-Holland, 1991.

[BBV91] A. Braccini, A. Del Bimbo, and E. Vicario. Interprocess communication dependency on network load. *IEEE Trans. on Software Engineering*, 17(4):357–369, 1991.

[BDBL91] D. A. Bradley, D. Dawson, N. C. Burd, and A. J. Loader. *Mechatronics.* Chapman and Hall, 1991.

[BDP95] E. Bertolissi, C. Daffara, and C. Preece. Rapid protocol development: a framework for fast protocol implementation. in preparation, 1995.

[Ben93] K. Bender, editor. *PROFIBUS; The Fieldbus for Industrial Automation.* Prentice Hall, 1993.

[BG76] M. C. Bonney and S. W. Gundry. Solutions to the constrained flowshop sequencing problem. *Operations Research Quarterly*, 24:869–883, 1976.

[BG91] M. Brill and U. Gramm. MMS : MAP application services for the manufacturing industry. *Computer Networks and ISDN Systems*, 21(5):357–380, 1991.

[BHS88] J. Browne, J. Harhen, and J. Shivnam. *Production Management System.* Addison-Wesley, 1988.

[BMK88] D. R. Boggs, J. C. Mogul, and C. A. Kent. WRL 88.4; measured capacity of an Ethernet: Myths and reality. Technical report, Digital Western Research Laboratory, 1988.

[Boc90] G. V. Bochmann. Protocol specification for OSI. *Computer Networks and ISDN Systems*, 18(3):167–184, 1990.

[BP96] E. Bertolissi and C. Preece. Assessment of real-time communication capabilities of network protocols in distributed systems. In *Proc. ITI '96*, pages 155–161, Pula, Croazia, June 1996.

[BWWH88] J. Billington, G. R. Wheeler, and M. C. Wilbur-Ham. PROTEAN: A high-level Petri net tool for the specification and verification of communication protocols. *IEEE Transaction on Software Engineering*, 14(3):301–315, 1988.

[Car88] A. Carrie. *Simulation of manufacturing Systems.* John Wiley, 1988.

[CB90] D. Chan and D.D. Bedworth. Design of a scheduling system for flexible manufacturing cells. *International Journal of Production Research*, 28(11):2037–2049, 1990.

[CG85] B. Croft and J. Gilmore. *RFC 951; Bootstrap Protocol (BOOTP).* Network Working Group, Menlo Park, September 1985.

[Cla90] D. Clayton. Experimental system for networking autostitchers in the factory. *SATRA Bulletin*, May 1990.

[CLV93] S. T. Chanson, A. A. F. Loureiro, and S. T. Vuong. On tools supporting the use of formal descriptions techniques in protocol development. *Computers Networks and ISDN Systems*, 25(7):723–739, 1993.

[Cod70] E. F. Codd. A realtional model of data for large shared data banks. *Communications of the ACM*, 13(6), June 1970.

[Cul96] A. G. Cullis. Sheffield Industrial Project Scheme; Electrostatic Deposition, March 1996. Second Year Industrial Project, Dept. of Engineering, University of Sheffield.

[DB74] J. M. Van Deman and K. R. Baker. Minimizing mean flow time in flowshop with no intermediate queues. *AIIE Trans.*, 6:28–34, 1974.

[Did89] Festo Didactic. Basi di data communication, 1989. course notes in Italian.

[Dro93] R. Droms. *RFC 1531; Dynamic Host Configuration Protocol.* Network Working Group, Menlo Park, October 1993.

[DVDR87] G. Doumeingts, B. Vallespir, DF. Darricau, and M. Roboam. Design methodology for advanced manufacturing systems. *Computers in Industry*, 9(4):271–296, 1987.

[EK93] M. Ettl and U. Klehmet. A performance comparison between the fieldbus protocol standards Profibus and FIP. *IFIP Transcation A: Computer Science and Technology*, 39:245–258, 1993.

[Fla96] D. Flanagan. *Java in a Nutshell.* O'Reilly & Associates, Inc, 1996.

[For96] Mits Forum. Maritime information technology standard. available from: http://www.itk.unit.no/SINTEF/MITS/mits.html, 1996.

[FR88] B. L. Foote and A. Ravindran. Production planning and scheduling. *Computers & Industrial Engineering*, 15:129–138, 1988.

[Fre88] J. R. Freer. *Computer Communications and Networks.* Pitman, London, UK, 1988.

[Gal95] B. O. Gallmeister. *POSIX.4; Programming for the Real World.* O'Reilly & Associates Inc., 1995.

[Gen87] General Motors. *Manufacturing Automation Protocol - Version 3.0 Implementation Release*, 1987.

[GM95] J. Gosling and H. McGilton. The java language environment; a white paper. http://java.sun.com/docs/langspec-draft5.1.Z, October 1995.

[Gou93] M. G. Gouda. Protocol verification made simple: a tutorial. *Computer networks and ISDN Systems*, 25(9):969–980, 1993.

[GR93] R. Gangadharan and C. Rajendran. Heuristic algorithms for scheduling in the no-wait flowshop. *International Journal of Production Economics*, 32(3):285–290, 1993.

[Gru87] R. J. Gruber. In *SID International Symposium Digest of Technical Papers*, page 272, Palisades, New York, 1987.

[Hal92] F. Halsaall. *Data Communications, Computer Networks and Open Systems.* Addison Wesley, 1992.

[Har85] P. Hardy. *Introducing Data Communication Protocols.* NCC Publications, Manchester, UK, 1985.

[Hoa74] C. A. R. Hoare. Monitors: an operating system structuring concept. *Communications of the ACM*, 17(10):549–557, October 1974.

[HR91] C. M. Harmonosky and S. F. Robohn. Real-time scheduling in computer integrated manufacturing; a review of recent research. *Int. J. Computer Integrated Manufacturing*, 4(6):331–340, 1991.

[HS89] S. Heatley and D. Stokesberry. Analysis of transport measurements over a local area network. *IEEE Comm. Mag.*, 27(6):16–22, June 1989.

[HS96] N. G. Hall and C. Sriskndarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, 1996.

[IEE85a] IEEE, Long Beach, California. *CSMA/CD Access Method and Physical Layer Specifications, IEEE Standard 802.3*, 1985.

[IEE85b] IEEE, Long Beach, California. *Token Ring Access Method and Physiscal Layer Specifications: IEEE Standard 802.5*, 1985.

[IEE95a] IEEE. *POSIX.1; System Application Program Interface - Amendment 2: Threads Extension (C language)*, 1995. Also ISO/IEC 9945-1:1990c.

[IEE95b] IEEE. *POSIX.1b; Application Program Interface - Real Time Extensions*, 1995. Draft Version.

[IIK+91] H. Ichikawa, M. Itoh, J. Kato, A. Takura, and M. Shibasaki. SDE: Incremental specification and development of communication software. *IEEE Transactions on Computers*, 40(4):553–561, 1991.

[Int84] International Organization for Standardization, Geneva, Switzerland. *Open System Interconnection: Basic Reference Model, ISO 7498*, 1984.

[JP94] M. J. Jubb and A. Purvis. Analysis of network protocol performance in the context of multi-workstations parallel distributed applications. *Microprocessing and Micropogramming*, 40(10-12):807–810, 1994.

[JS97] P. Jain and D. C. Schmidt. Experiences converting a C++ communication software framework to Java. *C++ Report Magazine*, January 1997.

[Ken83] W. Kent. A simple guide to five normal forms in relational database theory. *Communications of the ACM*, 26(2):120–125, 1983.

[KS80] J. R. King and A. S. Spachis. Heuristics for shopfloor scheduling. *International Journal of Production Research*, 18:343–357, 1980.

[KSN88] M. G. Ketchem, J. M. Smith, and B. O. Nnaji. An integrated data model for CIM planning and control. In *Proceeding of the Int. Conf. on Computer Integrated Manufacturing*, pages 338–342, New York, May 1988.

[LB96] K. Lai and M. Baker. A performance comparison of UNIX operating systems on the Pentium. In *Proc. USENIX Technical Conference*, January 1996.

[Lea96] D. Lea. *Concurrent Java: Design Principles and Patterns.* Addison-Wesley, 1996.

[Liu89] M. T. Liu. Protocol engineering. *Advances in Computers*, 29:79–195, 1989.

[MC87] S. A. Melnyk and P. L. Carter. *Production Activity Control; A practical Guide.* Don Jones-Irwin, 1987.

[McC88] J. McConnell. *Internetworking Computer Systems.* Prentice-Hall, 1988.

[McM93] C. McMahon. *CADCAM, from principles to practice.* Addison-Wesley, 1993.

[Mil78] R. G. Miller, editor. *Manual of shoemaking, produced by the Training Department, Clarks Limited.* (s.l.), C & J Clark Ltd, second edition, 1978.

[ML93] B. L. MacCarthy and J. Liu. Addressing the gap in scheduling research; a review of optimization and heuristic methods in production scheduling. *Int. J. Prod. Res.*, 31(1):59–79, 1993.

[MM88] D. A. Marca and C. L. McGowan. *SADT - Structured Analysis and Design Technique.* Prentice-Hall, 1988.

[MMCC95] A. Marsal, A. M. Mainch, M. D. De Castellar, and J. Cot. Study of the dissipation of the electrostatic charge of leather. *Journal of the Society of Leather Technologists and Chemists*, 79(4):115–118, 1995.

[Nag84] J. Nagle. *RFC 896: Congestion Control in IP/TCP Internetworks.* Network Working Group, 1984.

[NHAK89] S. Nakamura, H. Hirabayashi, J. Araya, and N. Koitabashi. U.S. Patent 4851960, 1989.

[Nil96a] K. Nilsen. Embedded real-time development in the Java language. In *Embedded Systems Conference East*, Boston, April 1996.

[Nil96b] K. Nilsen. Issues in the design and implementation of real-time Java. *Java Developer's Journal*, 1996.

[NS91] S. J. Noronha and V. V. S. Sarma. Knowledge-based approaches for scheduling problems: a survey. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):160–171, 1991.

[OP92] G. Olsson and G. Piani. *Computer System for Automation and Control*, chapter 10, pages 317–368. Prentice Hall, 1992.

[Peh90] B. Pehrson. Protocol verification for OSI. *Computer Networks and ISDN Systems*, 18(3):185–201, 1990.

[PGTNW97] T. A. Probsting, P. Bridges G. Townsend, J. H. H. Newsham, and S. Watterson. Toba: Java for Applications, A way Ahead of Time (WAT) Compiler. In *Proceedings of the 3rd Conference on OO Technologies and Systems*, 1997.

[Pim90] J. R. Pimentel. *Communication Networks for Manufacturing.* Prentice-Hall, 1990.

[Pin95] J. Pinto. Fieldbus: an international view. *Control and Instrumentation*, 27(2):48–49, February 1995.

[Pos80] J. B. Postel. *RFC 768, User Datagram Protocol.* USC/Information Sciences Institute, 1980.

[Pos81] J. B. Postel. *RFC 793: Transmission Control Protocol; DARPA Internet Program Protocol Specification.* USC/Information Sciences Institute, 1981.

[Pos82] J. Postel. *RFC 821: Simple Mail Transfer Protocol.* USC/Information Sciences Institute, 1982.

[PR83] RFC 854: J. Postel and J.K. Reynolds. *Telnet Protocol specification.* Network Working Group, 1983.

[PR85] J. Postel and J.K. Reynolds. *RFC 959: File Transfer Protocol.* Network Working Group, 1985.

[Pri91] D. Price. InterCIM standards agreed for all the current CAD/CAM requirements. *SATRA Bulletin*, January 1991.

[Pro97] Jolt Project. Kaffe: A free virtual machine to run Java code. available from http://www.kaffe.org/, 1997.

[PS91] R. L. Probert and K. Saleh. Synthesis of communication protocols: Survey and assessment. *IEEE Transaction on Computers*, 40(4):468–475, 1991.

[PVRS95] E. Pozzetti, V. Vetland, J. Rolia, and G. Serazzi. Characterizing the resources demands of TCP/IP. In *Proc. Int. Conf. on High-Performance Computing and Networking*, Milan, Italy, 3-5 May 1995.

[PW80] S. S. Panwalkar and C. R. Woollam. Ordered flowshop problems with no in-process waiting: further results. *J. Opns. Res. Soc.*, 31:1039–1043, 1980.

[Raj94] C. Rajendran. A no-wait flowshop scheduling heuristic to minimize makespan. *J. Opl. Res. Soc.*, 45(4):472–478, 1994.

[Ran83] P. Ranky. *The design and the operation of FMS.* North-Holland Publishing Company, 1983.

[RB86] J. R. Rumsey and D. Bennewitz. Ion printing technology. *Journal of Imaging Technology*, 12(3):144–151, June 1986.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Englewood Cliffs : Prentice Hall,, 1991.

[RC90] C. Rajendran and D. Chauhuri. Heuristic algorithms for continuous flow-shop problem. *Naval Research Logistic*, 37(5):695–705, 1990.

[RD89] M.G. Rodd and F. Deravi. *Communication Systems for Industrial Automation.* Prentice Hall, 1989.

[Roe95] D. Roeder. Fieldbus: Handicap hurdles. *Control and Instrumentation*, 27(8):27–28, August 1995.

[Sar93] P. Sargent. Inherently flexible cell communications: A rewiew. *Computer Integrated Manufacturing Systems*, 6(4), 1993.

[Sch75] R. M. Schaffert. *Electrophotography.* Focal Press, London, 1975.

[She92] L. B. Shein. *Electrophotography and Development Physics.* Springer-Verlang, second edition, 1992.

[Sol92] K. Sollins. *RFC 1350: The TFTP protocol (Revision 2).* Network Working Group, 1992.

[SR85] M. Stonebaker and L. Rowe. The design of POSTGRES. In *ACM-SIGMOD Conference of Management of Data*, Washington, DC, May 1985.

[SSM90] T. Suzuki, S. M. Shatz, and T. Murata. A protocol modeling and verification approach based on a specific language and Petri nets. *IEEE Transaction on Software Engineering*, 16(5):523–536, 1990.

[Svo90] L. Svobodova. Measured performance of transport service in LANs. *Comput. Networks ISDN Syst.*, 18(1):31–35, 1990.

[Tan81] A. S. Tanenbaum. Networks protocols. *Computing Surveys*, 13(4):453–487, December 1981.

[Tan88] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall, 1988.

[Top93] S. K. Topis. *Investigation of the Electrical and Mechanical Requirements for the Automation of a Process in Flexible Material Manufacture.* PhD thesis, University of Durham, Durham, UK, 1993.

[Tou89] N. R. Tout. *Investigation of the Processes Required for the Automation of Stitchmarking in Shoe Manufacture.* PhD thesis, University of Durham, Durham, UK, 1989.

[Wat94] R. C. Waterbury. Fieldbus forges ahead regardless. *Intech*, 41(1):38–40, January 1994.

[Wea88] A. Weatherall. *Computer Integrated Manufacturing.* Butterworth & Co Ltd, 1988.

[WH86] M. K. Wheatley and M. Hateley. Prediction of transport protocol performance through simulation. In *Siccom Conf. Proc.*, 1986.

[Wil26] J. A. Wilson. Effect of splitting on the tensile stregth of leather. *Industrial and Engineering Chemistry*, 18:312–313, 1926.

[Wu92] B. Wu. *Manufacturing System Design and Analysis.* Chapman and Hall, 1992.

[YC95] A. Yu and J. Chen. *The POSTGRES95 User Manual.* University of California Berkeley, 1995.