# A Distributed Computer Control and Information Exchange System

**L Petropoulakis**
Industrial Control Centre
Department of Electronic and Electrical Engineering
University of Strathclyde
50 George Street,
Glasgow G1 1QE
UNITED KINGDOM

**Abstract:** Controlling manufacturing systems requires the processing and exchange of large amounts of data. The magnitude of the data is often such that real-time communication cannot always be exercised. In domains where safety procedures need to be observed and where emergency signals are of high priority such considerations are particularly important. Often it is necessary to restrict the volume of information exchange to a minimum in order to accommodate hardware and protocol limitations. The latter necessitates keeping the various activities within the manufacturing domain independent of one another. This practical requirement, however, is contrary to the operational needs of modern manufacturing plants where materials, machine tools, conveyors, robots, handling systems, etc. must exhibit good synchronisation for effective flow control

**Dr Petropoulakis** was born in Athens, Greece. He graduated from the University of Salford with a first class honours degree in Aeronautical Engineering in 1982, and he obtained his Ph.D in Control Engineering in 1986 at the same university.

Dr Petropoulakis worked as a research fellow in Robotics and Vision systems at the Department of Artificial Intelligence, University of Edinburgh for four years before joining the Electronic and Electrical Engineering Department of Strathclyde University as a lecturer in Control and Robotics.

Dr. Petropoulakis' interests are in robotic systems and manufacturing, control architectures for robotic systems, neural-net technology for fault diagnosis, and generic manipulation procedures.

## 1.0 Introduction

Modern manufacturing networks utilise standard concepts such as star, ring or bus topology for ommunicating information. These methods are well known and have been in use for several years. However, the underlying architectural design structure which governs the information exchange in each domain can differ widely. This affects the functionality of the system and results in different degrees of efficiency. Some of these functionality differences can be found in [1], [2], [3]. In real-time communications speed is an essential element of data transmission particularly where large data transfers and/or critical time-delays are involved [5]. In view of hardware limitations exchange of information is kept to the minimum sustainable volume. In recent years higher rates of transmission have permitted increasing amount of information, but the improvements have already been matched by the increased demand. It is likely that this situation continues for many years as the demand for faster communications increases in all fields of engineering. It is therefore essential that the best possible management of resources can be made available at all times. There are two obvious methods of achieving this: a) Better use of hardware resources and b) Better control of the transmitted data. This paper outlines the main features of such a communication system.

The architecture is being developed for information exchange purposes in a robot assembly cell [4], but certain parts of this system have already been tested in simulation in other areas within the broader spectrum of manufacturing. The system has a modular and hierarchical design with an architecture which is characterised by its flexibility, portability and incremental expandability. The design is suitable for use on a combination of bus and ring (or star) topologies and it makes provision for some additional features in order to ease congestion of data flow and thus facilitate faster exchange of information, whilst it offers sufficient flexibility to suit particular needs. The design can also take into consideration modern technology applications where a multitude of data can be transmitted (at different rates) on the same network.

## 2.0 System Overview

In its simplest form the system consists of a central highway where all *stations* are connected. Each *station* can be an individual process within the manufacturing domain, or a *unit* comprising several sub-processes with a common aim. Sub-processes within a *unit* are linked together with a ring (or star) topology and are connected to the central highway via a switching computer. An overview of the overall system arrangement is shown in Figure 2.0.1.

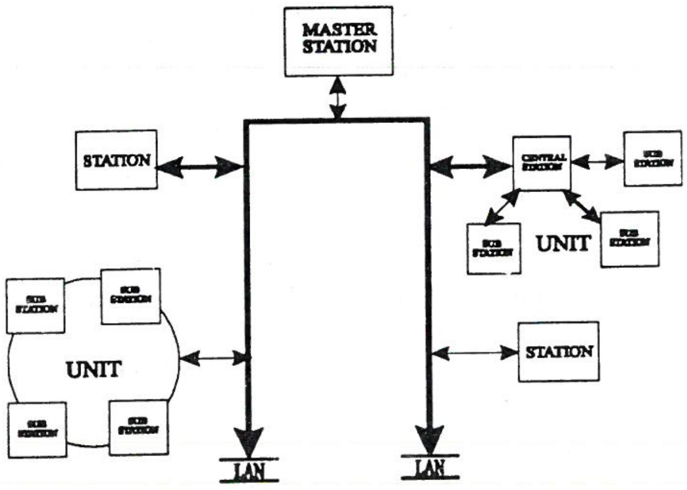

**Figure 2.0.1 Architectural Overview**

Each *station* or *unit* is associated with a **dynamic** database controlled through a dedicated processor which links the database to its *station* whilst, at the same time, it provides a direct link to the central highway system. The arrangement enables direct access of the database from other parts of the network. Although not currently used within the present design, the theoretical implication here is that of a multi-bus multi-processing system which can provide *stations* with the ability of transmitting and receiving information simultaneously.

This design offers a number of advantages but also presents some synchronisation difficulties. The main problem is to ensure that databases are updated correctly and that the requested information is supplied at the appropriate time. This is achieved mainly by taking into consideration the practical aims of a process.

In theory any *station* within the network can access the database of any other *station* at any time. Within any domain, however, the information generated by a given *station*, will only be of interest to a limited and (known) number of *client_stations*. This fact is used in the architectural design to control and update the information databases within the system. Hence, the information generated by any *station* is arranged in the database in sectors. Each of these sectors is created according to two main criteria: a) the information content and b) the known recipients of this information. When all *clients* have interrogated the database and received the information, this particular sector of the database can then be updated.

It is obvious that, in this arrangement, each *station* requires prior knowledge of its *clients*. This information would normally become available upon system instantiation through *information* or *configuration elements* assigned to the *stations* and/or *master stations* of central highways.

*Configuration elements (C-E's)* are structures within databases indicating the overall structure of a particular section of the design. Their advantage is that they provide a suitable medium through which the design is updated when *stations* or individual sub-processes are added to, or taken away from, the system. Then a simple polling procedure ensures that all devices are operating as expected. Should there be a change this is passed to the *C-E* controller.

The design overview described so far constitutes a very basic arrangement of a Distributed Control System (DCS) to deal with a central highway of data flow. Unlike other designs, this protocol dictates that under normal operating conditions *station* communication is effected through their respective databases. Only on occasions of high priority or emergency direct communication is permissible. This approach permits an easier and more flexible protocol of communication than it would otherwise be possible.

## 2.1 The Design

The initial requirement was for a data-handling protocol which could be developed to meet the following criteria:

a) Understand and communicate in natural language.

b) Process information as required.

c) Increase speed of communication between devices.

d) Provide a data exchange protocol which could be adapted to different environments and operational conditions.

The fundamental system design consists of a number of basic structures.

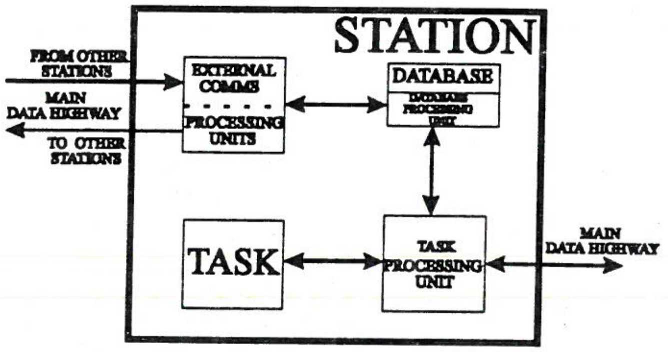

**Figure 2.1.1 A Station Structure**

1. **Stations**: In the general case the architecture of a single *station* will comprise a number of processing units depending on the complexity of the station's task. Central to the architecture is the station's database which is connected both to the main task processor and the central highway system through at least one, dedicated processor which controls information flow from and to the database (Figure 2.1.1). Figure 2.1.1 makes explicit the fact that the architecture permits more than one way to access the station's database. The design details of this process are explained in the relevant section for databases.

2. **Units**: These are simply collections of stations with a common aim (e.g. production of an assembly component). *Units* have a common database which is updated from the databases of the individual sub-stations. External access to the common database is permitted in exactly the same way as described for a *station*. In this case however, access to the databases of individual *sub-stations* within a *unit* is permitted only to components internal to that *unit*. The number of stations comprising a *unit* is unimportant.
3. **Databases**: These are dynamic structures which contain all the information essential to the correct functioning of a given station within the domain. Typically, databases are sectioned not only according to their information content, but also according to known servers and immediate clients of this information. This subdivision is effected from the contents of the *configuration_element* (C-E) structure. Each *station* contains its own C-E structure as part of its database. C-E structures provide a consistent way for updating system knowledge at both local and global level.
The C-E structure of a *station*'s database contains all knowledge regarding the various components which exist within the *station*, their respective tasks, their known servers and immediate clients. This classification permits a number of tasks to be performed with relative ease. For example, the addition or removal of a component and its task can be registered and the information communicated. Also the recipients of particular information can be notified, or their enquiries can be registered. In its simplest form a *C-E* database may resemble the following table:

**TABLE 2.1.1 UNIT 4 - STATIONS**

| Item | Function | Location | Server | Client |
|---|---|---|---|---|
| 1 Puma1 | Pick,Place | U4:S5:I1:: | U3:S2:I2:C1:: | U4:S5:I3:: |
| 2 Adept1 | Insertion | U4:S5:I2:: | U4:S5:I3:: | S15:I8:C2:: |
| 3 Lathe | Turning | U4:S5:I3:: | U4:S5:I1:: | U4:S5:I2:: |
| 4 Conveyor1 | Transport | U3:-:U4:S5:I5:: | U3:S2:I2:: | U4:S5:I1:: |
| 5 Conveyor2 | Transport | U4:-:U5:S5:I6:: | U4:S5:I2:: | S15:I8:: |

Table 2.1.1 contains information regarding the items, location, tasks, servers and clients of *unit4*, *station5*. The meaning of these abbreviated data is: U4=unit4, S5=station5, I3=item3,:=

data_ separator, ::= end_of_data, :-: = connection.

Hence, the first line under the heading **Server** (U3:S2:I2:C1::) means the following: Server is unit3:station2:item2:Carrier_is_conveyor1 (the data described here concern *item1*). The symbolism U3:-:U4 indicates a pipe (or connection) between *units* and signifies the beginning and end of this connection. Connections are normally associated with direct physical connections. C-E structures can be expanded as needed to contain more information (e.g. time schedules, time-out events, additional and/or alternative routes, etc). The present mechanism to do so relies on a simple interpreter of the symbolism used, which is consulted at boot time. This is held at *the master station* (Figure 2.0.1).

The master C-E structure, containing all the information regarding the overall system, is also held at the *master station* and it is propagated to the individual units, stations and sub-stations. In this way, information regarding the communication protocol needs only be carried out at global level. Note that the *master station* has no real control or explicit knowledge of the information exchanged between stations.

Using this configuration, the information exchange protocol can be considerably simplified since each item needs only communicate with known clients and servers. In other words the communication is swift and focussed. Additional items, stations, units can be added or taken away as required, provided that the database is updated accordingly. Should a particular *item* in the station carries out more than one task, the *item* is contained in the database as an additional quantity. Information sent to this *item* is simply queued by the database's mailbox system.

For this methodology to work, information in database entries must be arranged according to client or server arrangement, and according to the corresponding data_items. For each data_item a check_box exists to indicate whether the data have been retrieved (in the case of a client) or updated (in the case of a server). If not already in use, this is the only additional overhead on existing database structures currently employed in industrial environments, and it is only needed for the information to be exchanged between stations. The language used to describe actions and information within this structure can be altered at the *master station* level and propagated to the lower levels. This would be specific to the application, but the presented approach is not language-dependent.

The system operates on a client-server basis. Central to this operation is the database processor, which performs a number of specific tasks:

1. It examines the check_box of all the clients (and servers), for each record of the database.
2. It informs clients of new updates.
3. It permits updating of a record when all clients have acquired the information.
4. It requests information from servers when essential.

Clients of the database do need to inform the processing unit of their requests. They simply access the database and indicate (by writing to the corresponding check_box) that they have done so. There are cases where new information may not be of interest to *client*. In such cases *clients* still need to inform the server that the information has been received. Similarly, following a request by the processing unit, servers may update the database directly, and indicate their actions to their *clients*. This logging procedure requires minimum supervision and it permits a theoretical simultaneous access of the database by several clients and servers. In practice this can only be achieved if the database is "served" by a number of processing units.

Figure 2.1.1 indicates that direct communication between stations and tasks is also possible but, in the current implementation, this is only permitted in exceptional circumstances (i.e. high priority, or emergency signals) where essential.

## 3.0 The Initial Tests

The tests carried out so far involved a real implementation of limited scope, and a simulation of a manufacturing domain using the discrete event simulator SIMPLE$^{++}$.

The initial tests were carried out on an assembly system which comprises two robots equipped with simple tactile sensors, variable range (0.1 - 2m) digitally controlled sonars, photocell arrangements for the grippers, and purposely developed robot control software. The system runs under UNIX on a Sun 4 Sparc station IPC. This is connected to two PCs, also running under UNIX, using a LAN 10-baseT thin ethernet. Two internal Data Acquisition cards are used in the PCs (equipped with D/A and A/D converters) for obtaining information from the sensors. The PCs house the robot control software, pass instructions to the robots through the serial ports, provide the processing power needed for simple data sensor analysis, communicate to and receive information from the IPC. A direct link between the PC's using the parallel ports for simple signal exchanges is used as an emergency communication link.

Limited trials involving a vision system have also been carried out. The overall arrangement has been used so far to test simple assemblies and examine suitable strategies for assembling parts. Various constraints are imposed to simulate conditions likely to be encountered in assembly domains. These include part acquisition and part placement constraints, constraints occurring due to a large number of objects within the assembly domain, constraints due to motions of two co-operating robots, and constraints due to lack of one or more degrees of freedom.

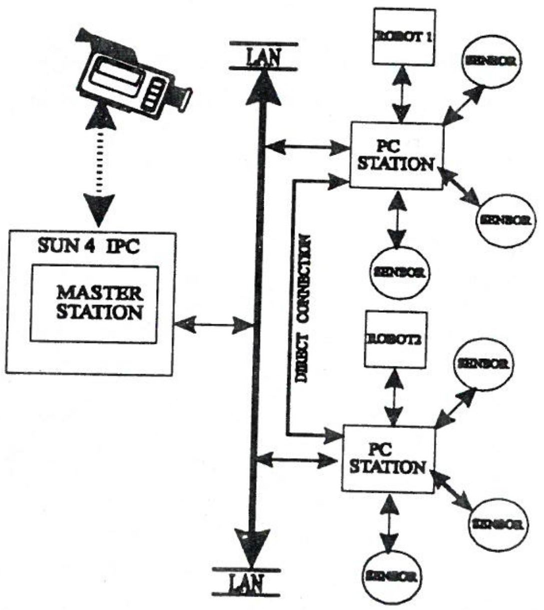

**Figure 3.0.1. Implemented System**

Within this modest arrangement the use of sensors is governed dynamically by the motion of the robots. Hence, a limited number of sensors is being used to assess a multitude of different conditions. This can easily create conflicting demands made on the sensors, and careful analysis and communication is needed. An overview of the implemented system is shown in Figure 3.0.1.

This arrangement was used to simulate the architecture described in Section 2. Each PC has its own database. The databases are built and updated according to existing knowledge and information supplied by the sensors at run time. Every database is available to all processing units, but it only stores information regarding its own localised domain. The information is supplied collectively by all sensors (including the vision system). The workstation has the role of providing supervisory control, controlling the actions of the camera, and provides a reasoning module where sensor information is analysed and interpreted. The camera is on a tilt and pan mechanism enabling additional viewpoints (it has also been tested mounted on one of the robot arms).

In the tests, the camera is used as the main global source of information, but its results are supplemented by the localised sensors and a priori knowledge. As all the sensors are used dynamically, the resulting database is reasonably rich in information regarding the state of the assembly. This information is initially acquired on the workstation and then communicated to the PC's.

The low-level protocol of the message exchange mechanism [6] is based on the International Telegraph & Telephone Consultative Committee (CCITT). Messages are constructed of a number of fields (the first five are common to all) depending on content and complexity. They are:

a) The START indicator
b) The PRIORITY flag
c) Message ID (DATA, REQ, ACK, or ERR)
d) Device ID
e) Message LENGTH

In this way, message handlers (e.g. mailboxes) do not need to know the actual data that are communicated between devices. Their capabilities extend to the degree of examining the signal priority and handling it accordingly, suspending other communications if necessary. In this test-bed implementation the approach was that high priority signals between the PC's were transmitted through parallel ports. This was simply done to help increase the speed of the signals in this arrangement.

For most tests the procedure appears to run satisfactorily given the hardware limitations of this system. There are certain advantages in the arrangement described so far, particularly since the sensors are used in a dynamic manner. For example, a vision system provides information not only about the particular point of interest at the time, but also about a wider area within the domain. This information can be stored and used as required. It has the advantage of better exploiting the existing resources, and of faster transferring information (raw data are only processed once).

Here are some of the difficulties encountered :

a) Ensuring an up-to-date state of the database. This can be difficult to achieve in an environment with limited resources. If parts change position in the environment, the database must be updated immediately and all actions based on previous information must be checked. One way of shunning this problem is to ensure that parts situations described in databases will not change before actions which are related to them have been completed. This may be difficult to ensure in a complex environment.

b) Using the aforementioned method for evaluating and storing information resulted in rather large (compared to the task involved) dynamic databases which were not always required. In practice only about 10% (at best) of the dynamic databases created were re-used in some way. This may well suggest that use of sensors as needed may be not such a wasteful process after all. However, the tasks involved were rather unrepresentative of large manufacturing plants and the findings may be a reflection of this very fact. This action could not really be simulated in the discrete event system in a fully unbiased way.

c) The third difficulty encountered with this approach was that the level of communication got considerably high with dynamic use of sensors. This could not always be handled by the modest arrangement in the practical test. In the simulation, however, the use of multi-processing units appears to provide the answer to the problem of large amounts of data exchange being involved.

## 4.0 Conclusions

The results of the practical implementation and of the simulation of a manufacturing process were largely in unison and as expected. As the discrete simulation package has real-time capabilities, we were able to make a realistic inference regarding the difficulties likely to be encountered in a large domain. There is little problem in expanding this type of communication architecture to levels of complexity comprising several highways of data flow since there should not really be any dependency between data representation and data exchange. Assuming this to be true, in much the same way as individual *stations* control their flow of information, individual highways can also be connected to provide an expanded network of information. In view of the difficulties met with in updating the databases and in ensuring the reliability and correct timing of the information, it is felt that additional control may be required for guaranteeing absolute reliability. The large number of unused records is not a real practical problem especially in a distributed system.

The system architecture is in unison with current trends of Functional Software Architectures (FAS) and peer-to-peer network concepts. It uses however multiple channels of communication, which is not a feature yet widely used in industrial domains.

## REFERENCES

1. KOJIMA, F. and KOIKE, T., **Advanced Configuration Tools for DCS**. ISA TRANS., Vol. 30, No. 2, 1991, pp 25-32.

2. TAKAYULI, M. et al, **Engineering Tool for CIEMAC**, TOSHIBA REVIEW, Vol. 44, No. 6, 1989, pp. 470--473 .

3. POPOVICH, D. and BHATAS, V. P. , **Distributed Computer Control for Industrial Automation**, MARCEL DEKKER INC., New York and Basel, 1990.

4. PETROPOULAKIS, L. , **Robot Assembly: An Architecture to Promote Autonomy**, Proc. of 20th Int. Conf. on Industrial Electronics, Control and Instrumentation (IECON 94), Vol. 2, Bologna , Italy, 1994, pp.952-957.

5. CLEMENTE, G. , CONGIU, S. and MORO, M., **Communication Software for Concurrent Real-Time Control**, Proc. of IECON 92, San Diego, CA,1992.

6. GALSWORTHY, P., **Design and Development of A Real-Time Multi-Processing Control System,** Internal Report submitted for the degree of B.Eng. in Computer & Electronic Systems, University of Strathclyde., April 1995.