Aston University

Some pages of this thesis may have been removed for copyright restrictions.

If you have discovered material in AURA which is unlawful e.g. breaches copyright, (either yours or that of a third party) or any other law, including but not limited to those relating to patent, trademark, confidentiality, data protection, obscenity, defamation, libel, then please read our Takedown Policy and contact the service immediately

A STRUCTURAL METHOD FOR THE DESIGN OF FAULT TOLERANT DISTRIBUTED CONTROL SYSTEMS.

Mahendra Magan Patel, M.Sc.

Submitted for the degree of **Doctor of Philosophy.**

The University of Aston in Birmingham

September 1988.

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior, written consent.

THE UNIVERSITY OF ASTON IN BIRMINGHAM

A Structural Method for the Design of Fault Tolerant Distributed Control Systems.

Mahendra Magan Patel, M.Sc

Submitted for the degree of **Doctor of Philosophy** 1988.

Summary.

Distributed digital control systems provide alternatives to conventional, centralised digital control systems. Typically, a modern distributed control system will comprise a multi-processor or network of processors, a communications network, an associated set of sensors and actuators, and the systems and applications software. This thesis addresses the problem of how to design robust decentralised control systems, such as those used to control event-driven, real-time processes in time-critical environments.

Emphasis is placed on studying the dynamical behaviour of a system and identifying ways of partitioning the system so that it may be controlled in a distributed manner. A structural partitioning technique is adopted which makes use of natural physical sub-processes in the system, which are then mapped into the software processes to control the system. However, communications are required between the processes because of the disjoint nature of the distributed (i.e. partitioned) state of the physical system.

The structural partitioning technique, and recent developments in the theory of potential controllability and observability of a system, are the basis for the design of controllers. In particular, the method is used to derive a decentralised estimate of the state vector for a continuous-time system. The work is also extended to derive a distributed estimate for a discrete-time system.

Emphasis is also given to the role of communications in the distributed control of processes and to the partitioning technique necessary to design distributed and decentralised systems with resilient structures. A method is presented for the systematic identification of necessary communications for distributed control. It is also shown that the structural partitions can be used directly in the design of software fault tolerant concurrent controllers. In particular, the structural partition can be used to identify the boundary of the conversation which can be used to protect a specific part of the system. In addition, for certain classes of system, the partitions can be used to identify processes which may be dynamically reconfigured in the event of a fault. These methods should be of use in the design of robust distributed systems.

Keywords: ***Dynamic reconfiguration, Fault tolerant software, Conversations, Necessary Communications, Structural Partitioning, Design Methods and Methodology.***

ACKNOWLEDGEMENTS.

There are a number of people who have helped during the course of this work and in the production of this thesis.

A special thanks are due to **Dr. Geoffrey F. Carpenter** and **Dr. Robert C. Johnson** for many stimulating discussions and their help in the preparation of this thesis. I would also like to thank the **S.E.R.C.** for their initial funding over two years and **Aston University** in the final year. Finally I am indebted to my supervisor **Dr. David J. Holding** without whose constant encouragement and searching questions, this work would not have been possible.

LIST OF CONTENTS.

LIST OF FIGURES.

CHAPTER 1

1.0 INTRODUCTION

1.1 INTRODUCTION

A typical distributed control system consists of a set of information acquisition or measurement units and a set of control actuation units (both interfaced to the physical system), together with a set of control units implementing a control strategy (decision processes). The physical distribution of the measurement and control interface is assumed to be matched to physical system requirements. The controllers will be distributed according to operational, economic, or geographic criteria, and will be linked to interface units and to other controllers by a communications network.

Two major factors have contributed to the rapid development of the theory of decentralised control in the last two decades. These are the need to accomplish computation in finite time and the increasing complexity of systems [1].

The reliability of a computer system decreases very rapidly with increasing complexity. Hence a method must be developed for handling large, complex problems. Ideally a functional decomposition technique must be used to make the design problem more manageable. From a practical point of view it is far easier to manage a set of small problems than a single large one.

Furthermore, functional decomposition leads to small understandable processes, which are more likely to be amenable to formal proof and may be candidates for concurrent execution. The main

disadvantage of the technique is that the sub-problems must be coordinated, as will be shown later. The pioneering work in this area was done by Kron [2] and by Dantzig and Wolfe [3], who first introduced the concepts of decomposition and coordination.

As an example of a decentralised control system consider the system of **Figure** 1.1 (reference 1). It consists of two cars which are connected by a spring. The problem is to move the system from a point A to a point B with minimum fuel cost. The two controllers are the drivers of the cars and they have access to only part of the information space, I_p. I_p consists of the characteristics of the two cars and of the spring between them and is partitioned into two sub-spaces, I^i_p (i=1,2), each of which contains the characteristics of the car (i) and the characteristics of the spring.

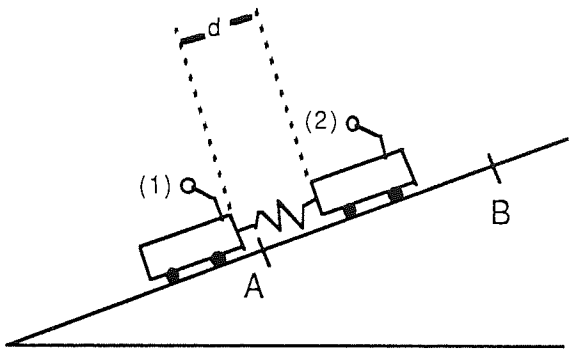


Figure 1.1 An example of a decentralised control system

It is clear that the two controllers arrive at conflicting solutions for the optimal spring length d. Driver (1) requires the spring to be in tension so that car (1) is pulled up by car (2), and driver (2) requires the spring to be in compression so that car (2) is pushed up by car (1). In each case the spring length desired by the controller is such that the fuel consumption of its car in its movement from A to B is a minimum.

This conflict between the sub-processes is typical in decentralised control. It arises because the information space common

to both sub-systems is processed independently by the sub-controllers. On the other hand, if properly coordinated, the two controllers can work concurrently, which can result in faster execution time.

Therefore, it has to be assumed that some adequate model of the system is available in which continuous-time properties are represented by linear mathematics. The concept of partitioning can then be applied methodically. However, large-scale linear systems present problems associated with rank determination, solubility and design. Controllability, observability and pole assignment can become difficult to resolve, even with the use of computer-aided methods, because the initial data itself cannot be specified with sufficient accuracy.

Because of these difficulties the design of complex linear systems involves qualitative or structural procedures without actually requiring explicit solutions [4]. This partitioning will lead to the problem of how to overlay a distributed control structure on a distributed system, that is, the problem of matching a type of controller to each partition and the identification of inter-partition communication necessary for total control. Each controller's software will comprise a local observer and controller function and a communication interface to other controllers. To make this decentralised control system robust it may be necessary to consider the software fault tolerance measures applicable to such a loosely coupled distributed system.

In the research presented in this thesis the model is transformed from a quantitative representation to a Boolean form and then mapped onto a digraph to facilitate analysis in the structural domain using graph theory. The structural partitioning technique is used to decompose the model into **cyclic** and **acyclic** sub-graphs (appendix A). The decomposition decouples each linear sub-system,

such that quantitative changes in the other linear sub-systems do not alter the sub-system's own dynamic properties [5].

The concurrent software processes required for control purposes are mapped from the decomposed physical system. One software process observes the behaviour of the linear sub-system and the second software process controls it using the estimate obtained from the observation process. Inter-process communication is required to achieve overall decentralised control since each controller interacts with the other's disjoint sub-state space as a result of the partition of the overall state space of the physical system. This implies that each controller has a naturally decomposed database by virtue of the structural partitioning. Control of the overall system in a decentralised manner requires necessary communications [6].

The next stage is to map these distributed communicating asynchronous processes and distributed databases onto a network of processors with a communication network. The mapping is done: (i) to achieve coordination and synchronisation; the structural partitioning technique employed, makes this straightforward. (ii) to achieve maximum parallelism with only the necessary communication between the controllers. (iii) to achieve optimum workload for each processor in the network.

The final stage of the design uses this generic partitioning technique with modern software design methods and control methods to achieve robustness in the design of a system. The decomposition of the linear system allows modern control techniques to be exploited and enhanced; the partitioning technique allows the designer to incorporate fault tolerant software for loosely coupled systems. Thus:

a) In the control aspect this generic partitioning technique can exploit the decentralised control strategy of Evans et al [7] to achieve decentralised observation for a linear continuous-time system. The

results are such that the observation processes are reliable in the presence of failures of other processes and lead to numerically simpler computation than the techniques of [8,9,10]. The sub-systems can be made completely autonomous by eliminating communications between them, and are hence more robust to other process failures, through the adoption of the **unknown observer** technique used in the centralised approach. The structural (generic) partitioning technique leads to decomposed sub-systems in which this can be adopted directly. This technique has been exploited in the discrete-time domain in the design of Kalman filters [11], which provide estimates in the presence of noise which is white, gaussian, and uncorrelated with the input and output of the system.

b) Large-scale systems present problems for development of software especially if the system is required to operate satisfactorily in the presence of faults. Techniques for the construction of fault tolerant software systems exist [12]. A fault tolerant system detects errors created by a fault and provides error recovery through exception mechanisms and algorithms which restore normal computations. These methods are based upon useful redundancy of design [13]. The generic partitioning technique can be exploited to generate concurrent processes which are atomic [14] and can form proper conversations [15]. It is noteworthy that these actions are identified at the decomposition stage rather than requiring identification after the software has been written [16].

The ultimate objective of the thesis is to derive a design methodology for distributed computer control of an event-driven, time-critical system. The terms 'method' and 'methodology' are often used interchangeably. In this thesis the term 'methodology' is used to mean a set of methods together with their theoretical basis.

1.2 SUMMARY of THESIS

Chapter 2 of the thesis sets out the objectives of the research presented here. It also identifies a number of tangible goals which were achieved to obtain the set objectives. It presents a survey of previous work in the field of decomposition. Chapter 3 and part of chapter 4 survey the previous work in the field of systematic identification of necessary communications, fault tolerant software, dynamic relocation and observation of decentralised systems to achieve the set objectives.

A crucial concept in the subsequent chapters of this thesis is the idea of a partitioning technique in order to support the design of distributed and decentralised control systems with resilient structures. A variety of methods are required to decompose a system at various stages of the design cycle. The literature survey indicated that the structural partitioning technique used by Evans et al [7] is ideal. This generic partitioning technique is shown later in the thesis to be applicable for decomposing a linear continuous-time system to generating concurrent software processes which are atomic [14] and can lead to the design of resilient software for a real-time system.

Chapter 3 summarises the use of graph theory in studies of controllability and observability for large-scale systems. This is then used to introduce the concept of necessary communications [6] in a decentralised system for the control of distributed processes. The necessary communication concept is extended to cater for the interaction of the processes.

Chapter 3 also introduces the conversation mechanism used to make a set of concurrent processes fault tolerant. The latter part of the chapter surveys the use of techniques in the dynamic reconfiguration of processes. The literature on fault tolerant software

and the techniques of dynamic reconfiguration processes are surveyed in order to incorporate resilient structures in software construction.

In chapter 4 the structural partitioning technique is used in conjunction with the decentralised control strategy of Evans et al [7] in order to achieve the decentralised observation of a linear continuous-time system. The observer is used to obtain an estimate of the state vector for each partitioned sub-system from its local measurements. This approach circumvents the need for a stability analysis of the complete interconnected sub-system and hence leads to a simpler numerical computation than that achieved by other researchers [8,9,10]. The next section of this chapter derives systematically the necessary communications required by a decentralised controller for observation. This work augments the necessary communication concept introduced in chapter 3. The necessary communications between the observation processes can be eliminated by adopting the **unknown observer** method. This leads to observation processes which are resilient to failures of other observation processes. Only basic matrix criteria are presented in this thesis for the unknown observer approach. The partitioning technique is further applied in the discrete-time domain for the first time to the Kalman filter method. This entails the use of several small order Kalman filters, communicating, to obtain an estimate of the complete state vector.

Chapter 5 presents an example in the the generic partitioning technique used to map the cyclic and acyclic sub-systems into the structure of control software. That is, it generates concurrent software processes which are shown to be atomic [14] and which can form proper conversations [15]. The design of distributed processes is pursued in the concurrent programming language Occam [17] and related to Petri net models. It is noteworthy that the identification of

atomic actions takes place at the decomposition stage rather than after the software has been written [16]. This identification highlights potential problems which may arise in protecting against interprocess communication failures; an interim solution is provided for protecting transactions with a distributed database in a time-critical real-time application. Finally, the thesis considers how to exploit the flexibility of a distributed system by providing dynamic reconfiguration. The possibility of reconfiguration is shown by using the directed graph technique and the potential controllability and observability criterion in the Boolean domain. The last section of this chapter describes the steps to be taken in the design of a distributed or decentralised control system.

Chapter 6 summarises the achievements of the research and draws a number of conclusions about these. Also in this chapter a number of areas for further research are suggested.

CHAPTER 2

2.0 AIMS AND OBJECTIVES OF THE RESEARCH

2.1 INTRODUCTION

The aim of this research is to develop a methodology for designing Distributed Control Systems using Computers or Microprocessors. The work includes studies of the relationship between the structure of control systems and that of the dynamic plant. Particular emphasis in the research is given to the role of **communications** in the distributed control of continuous processes, and to the **partitioning technique** necessary to support the design of Distributed and Decentralised Control Systems with resilient structures.

The problem of **coordination** and **synchronisation** in decentralised control was highlighted in chapter 1 due to decomposition. This problem arose from the fact that the overlapping portions of the information space of the overall system are processed independently by the sub-controllers. On the other hand the two controllers can work concurrently, which can result in faster execution time.

It has been assumed that the state equations are known and this research is aimed only at developing the software techniques for decentralised control. The first step in the software design is to **partition** a system into a number of reasonably sized tasks which may be able to run concurrently. The second step is to select a suitable **concurrent language** for design and implementation. The specification may place too great a demand on the language so that language extensions are required, for example in the design of systems which can tolerate failures in the component communicating

asynchronous processes. The second step in the design was investigated by a research colleague who developed a method for the design of fault tolerant software for loosely coupled distributed systems [16]. His work also showed a way of extending the concurrent language (**Occam** [17]) for communication failures [18].

It is unwise to partition arbitrarily. The designer needs to have design objectives and be able to answer the following questions: (1) **how are the tasks to be partitioned?; (2) will the partition alter the dynamical behaviour and the failure modes of the system?; (3) what effect does this partition have on the communication and computation systems?; (4) will it identify and produce a fail-safe design?**

Part of the **emphasis** in this thesis is placed on the **partitioning technique** necessary to support the design of distributed and decentralised control systems. The structured partitioning technique adopted allows the designer to answer all the questions posed earlier. This technique is generic to all levels of system design: from partitioning a linear system, (represented by mathematics) to the decomposition of software into processes, where resilient structures can also be identified.

This partitioning process makes use of **Cyclic** and **Acyclic** sub-systems (see appendix A) identified within a linear system. Use of this method allows the following four objectives to be satisfied:-

1) Estimation in continuous and discrete-time systems.

2) Systematic identification of the necessary communications for distributed control.

3) To show that the partitions are the natural side walls to the conversation used in fault tolerant design.

4) To show that dynamic reconfiguration is possible for certain classes of system.

2.2 PARTITIONING TECHNIQUES

Two major issues in system and software design are the processes of **partitioning** and **assignment**. Partitioning is a process of clustering disjoint subsets based upon the relative strength of relationship between elements [19]. Assignment is to allocate nodes of the system using the boundaries obtained by the above process according to some cost criterion.

Partitioning and assignment issues must be subjected to the following questions.

i) What criteria should be used to partition and assign?

ii) When should the partitioning occur?

iii) How long should a given assignment persist?

The first sub-section surveys the existing methods and considers: firstly, the decomposition (partitioning) technique embodied in a design method. In order to ascertain whether it satisfies the questions: (a) how are the tasks to be partitioned?; (b) What effect does this partition have on the communication and computation systems?, etc. Secondly, the design method. It discusses whether the method can be adapted directly for the aim of the research outlined in the first paragraph of this section.

The second sub-section deals with partitioning techniques which do not form part of any design methods. These are based on achieving specific performance goals. The third sub-section deals with partitioning based on the structure of the problem. The structured partitioning technique is a decomposition at an abstract level. Therefore, the existing software methods may be utilised at a lower level.

2.2.1 Software Design Methodology

2.2.1.1 Introduction

One of the aims of this thesis is to look at all the partitioning techniques necessary to support the development of a design methodology. This entailed examining several existing methods (for partitioning), which are widely used in the design of a system. The salient features of several of these techniques are presented below. All except **MASCOT** are fully illustrated in Software Engineering books [20,21]; **MASCOT** is illustrated fully in the official handbook of **MASCOT** [22].

Note: in any system design the software production goes through several stages of a development cycle. The methodology developed in this thesis does not account for all of the stages involved in the development cycle, like the **ISTAR** package [23]: which provides an integrated project support environment. This methodology will explicitly cater for the design of a system at an abstract level and provide a run time support for process failures: which overcomes the design faults (see section 3.3). However, it does not look at the maintenance and modification stages or any mathematical techniques for the specification stage, which will lead to a resilient design; for example like **Z** [24] or **Communicating Sequential Processes (CSP)** [25].

2.2.1.2 Program Methodologies

2.2.1.2.1 Functional Design Method (F.D.M.)

The **FDM** technique is one of the oldest and most widely

established, and arguably the first truly systematic method for program design [20,21]. **FDM** encompasses the techniques of **step-wise refinement** [26] and **structured programming** [27] i.e uses only **sequence, selection, repetition** and avoids the use of **goto** statements in coding. The technique starts with a single grand, statement of what the software is to do. This is refined using a **pseudo-language** (English imperitive sentences) written as sequences, with **if, while, etc** statement (structured statements). The design is refined until the required detail is achieved for coding in a suitable programming language. The choice of decomposition strategy has a major influence upon the quality of the resulting design.

There is no indication of how to decompose the problem; the approach is left to the designer who refines the software subjectively based on experience rather than some objective mechanical criterion. This can lead to solutions that are not maintainable or portable.

2.2.1.2.2 Data Flow Design (D.F.D)

This technique was developed by Yourdon, Myers and Constantine [28]. A design technique based upon Data stream anaysis called (variously) **Data flow design, Structured Analysis/Design** or sometimes **Transform Centered Design**. The analysis produces a **data flow diagram**, where each transformation that converts an input data flow into an output data flow is represented by a **bubble** and data flows by an **arc**. The method starts with a single, large bubble, which is broken up into smaller bubbles. This technique is widely used for real-time, time-critical control because most real-time systems are hierarchical in nature. The process of design is repeatable and less intuitive than the **FDM** technique and the decomposed modules will be maintainable since the design is based on **data coupling** [20,21].

DFD requires a great deal of inventive effort for a complex design with many data flows. The task of transforming the **DFD** into a hierarchy can be non-trival but becomes easier with experience if a sequential language is used. If a language like Occam [17] is used, which provides parallelism, then bubbles could be implemented as Occam processes with intercommunicating data between them. Therefore, a hierarchical transformation is not required. The general criticism of the method is that it relies on a principle which tends to ignore any structure in the problem and data domain.

2.2.1.2.3 Jackson Structured Programming (J.S.P)

Methods of software design based upon the structure of the data in the system have been proposed by Warnier, Jackson and others [29]. Michael Jackson is the foremost in this field, and developed a method well before publication in **1975** [30]. The basis of the technique is that the structure of a program can be derived from the structure of the files that the program acts upon or creates. The method uses a diagrammatic notation (a tree like notation which models the three structured constructs: sequence, repeat and selection) for the file and program structures and its own pseudo language. Using these notations as documentation, the method proceeds step-by-step from descriptions of the file structures to a design of a program structure.

The steps are [20]:

1. Draw a data structure diagram describing the structure of each of the files that the program uses.
2. Derive a single program structure diagram from the set of data structure diagrams.
3. Write down the elementary operations that the program will

have to carry out.

4. Associate the elementary operations with their appropiate positions in the program structure diagram.

5. Transform the program structure diagram into schematic logic.

This technique has been widely used [31]. This technique is distinct, rational, self-contained and well defined and produces code that is **modifiable, understandable and portable** because the algorithm is determined by the **data structure.** The main attribute of the technique is that decomposition guidelines are very instructive and therefore quality of design **converges**; several designers will come up with remarkably similar solutions to a given problem.

The main problem with the technique is in the minority of cases, where there may be incompatibility between the input and output structure. This is known as a structure clash. This is solved by introducing an intermediate program or a file; a process known as **program inversion**.

2.2.1.3 System Methodology

2.2.1.3.1 Jackson System Development (J.S.D.)

JSD was evolved from **JSP** as a methodology to apply to larger problems that required system solutions rather than a program solution [32]. It is important to note that **JSP** is not a subset of **JSD**; for example, in **JSP** the starting point is a full program specification, whereas in **JSD** the first major phase is to develop a system specification. Nevertheless the techniques share the same fundamental principles and characteristics: (i) both are process based; (ii) both

model the relevant aspect of the real-world; (iii) design is strictly in terms of sequential processes connected by sequential data streams. The functional details are left until a late stage; concurrency, databases etc. are seen as tools for implementation. **JSD** overcomes the disadvantage of **JSP** mentioned earlier.

To conduct **JSD** the following steps are applied [29]:

Entity action step. Entities (people, objects, or organisations that a system needs to produce or to use information) and actions (the events that occur in the real world that affect entities) are identified.

Entity structure step. Actions that affect each entity are ordered by time and represented by its Jacksons structured diagrams.

Initial model step. Entities and actions are represented as a process model; the connections between the model and the real world are defined.

Function step. Functions that correspond to defined actions are specified.

System timing step. Process scheduling characteristics are assessed and specified.

Implementation step. Hardware and software are specified as a design.

There is a large overhead involved in training the staff who are to use **JSD**. This is true of any good systematic technique, however the solution produced by this technique far out weighs any disadvantages because this technique will lead to the production of software systems that are modifiable, understandable and portable.

2.2.1.3.2 MASCOT

The **Modular Approach** to **Software Construction**

Operation and **Test** (**MASCOT**) was first introduced between **'71** and **'75** by Jackson and Simpson [33]. **MASCOT** is intended for use in the design and construction of software for real-time embedded computer systems. It is used throughout the design, construction and integration stages of software development and provides a framework for software maintenance throughout the life cycle of the system. **MASCOT** provides 3 features; (i) A graphical design method (the **Activity-channel-pool** or **ACP** diagram) which shows system structure and the flow of data within the system and which forms the basis for system construction, management and testing. (ii) A suite of construction software to combine the modules in an operational system. (iii) An executive program (usually called the **kernel**) which controls the interaction between modules of code (**activities** and **intercommunication data areas** or **IDA's**) at run-time, safeguards access to data, and provides monitoring facilities.

The main disadvantage of **MASCOT** is that it requires inventive effort by the designer at the decomposition stage (as in the case in **DFD** technique). Early versions of **MASCOT** were designed for a **multiprocessing environment** using direct **shared memory** and not for a **distributed processing environment**. **MASCOT 3** has to a large extent overcome these problems and is more effective for use with large systems [31].

2.2.1.3.3 Object-Oriented Programming

Object-Oriented programming is a method of decomposing a specification to produce a system design [20,34,35], which differs from the more traditional method of functional decomposition. The approach is based upon identifying a number of types of objects, where each object identifies some physical or conceptual object in the world

being modelled. Objects could include; a **communication protocol**, a **display window** or **icon**, file (data) **service** and **remote operations**. An **object** consists of **data** and a **set of operations** that may be performed upon the data. This approach allows **information-hiding** and **data abstraction**. New objects can **inherit** properties from existing objects. The interaction between objects is established at **run-time**. Therefore, Object-Oriented Programming provides a **flexible environment** in which product and application can be easily tailored to meet specific needs.

This has a similar **disadvantage** to **JSP** and **JSD**: i.e. a large overhead involved in training staff.

2.2.1.4 Assessment of software design methodology

The embedded partitioning technique in **FDM**, **DFD** and **MASCOT** are based on some arbitrary criterion defined by the designer. Fundamental questions such as how the partitions alter the dynamic behaviour and the failure mode cannot be answered in a systematic way. In **JSP**, **JSD** and **Object-Oriented Programming**, the partitioning technique is systematic with an objective rationale (reasons/principle). While these model some physical or conceptual object of the world which is appropriate for use in non-hazardous environments, like real-time distributed office systems and banking systems, it is not suitable for real-time control systems, such as nuclear plants, ballistic missiles, etc., because it does not model dynamic physical properties (i.e eigenvalues). Therefore fundamental questions cannot be answered. However, any of the systematic software methods above produces code that is modifiable, understandable and portable, therefore these should be used for coding the software.

2.2.2 Goal-oriented partitioning

2.2.2.1 Introduction

It is assumed here that the design effort has resulted in a refined and structured definition of the data flow and processing steps required to achieve desired actions or response from a system. Jensen et al [36] pose the following question: which or what criteria are to be used by the designers in partitioning a system. This section tries to answer that question.

A different strategy has to be used in partitioning depending on whether the design goals are growth, reliability or performance. Partitioning used to achieve the above goals in reference [19] are based on :

1) **Flow relationship: vertical** and **horizontal partitioning** [37]. **Vertical partitioning** is concerned with the separation of elements which have predecessor relationships stemming from data dependancy or mandatory control flow considerations. Two sets of processing can be **horizontally partitioned** if they can be executed concurrently or without regard to order. See appendix C for the description and rules of these techniques.

2) **Data access:** in this technique the notation of temporal order, and direction of data flow and control flow are ignored. Here, one concentrates upon the data space accessed, the partition will coincide that obtained with the horizontal and vertical partitioning technique. See appendix C for the rules of **data access partitioning**.

3) **Axiomatic partitioning:** defined by formal axiomatic decomposition techniques based on the structural composition of the system. D. L. Parnas [38] proposed an **axiomatic criterion** for modularity in which he states "each module should implement a design

decision and act to isolate and hide that decision from other modules." An example of this approach is the **Higher Order Software (HOS)** methodology [39].

2.2.2.2 Change and Growth

With time and various other reasons requirement specifications will change. The design must be such that adaptation to meet requirement specification change is easy. The major objective here is to **hide information** and to **isolate design decision**. One of the solutions is to confine the changes to a single module or, at worst, a small number of modules. The criteria here is to use a **what if** [38] analysis of the requirement specification, i.e. if this requirement changes, what aspect of the design is affected based on the **axiomatic criterion** proposed by Parnas [38].

2.2.2.3 Reliability

For high reliability, the design must adopt **fault prevention design** techniques and incorporate **fault-tolerant structures** into the design, so as to achieve a high degree of **robustness** in the resulting system. Fault prevention requires the use of constructs that are **demonstrably correct**; it requires the use of **structured software design** and **programming methodologies** to simplify the process of design analysis and verification. Fault-tolerant design involves the incorporation of structures to recognise errors, contain the scope of any damage, and to recover from the errors.

Damage containment can be achieved by the extensive use of **vertical** and **horizontal partitioning** to generate modules which are

small in terms of processing steps and the span of data used. The **fault-tolerant** aspect will be addressed later in this thesis in chapter 3; where the technique of **forward** and **backward error** has to be applied [12].

2.2.2.4 Performance

Distributed systems will use many n-processors distributed geographically or on a functional basis. However, using **n** processors, does not equate to **n** times the computational power as the communication media saturates, that is, the throughput decreases incrementally with increasing number of processors, if the processes are not assigned to processors optimally in the distributed system. The goal is to maximize the concurrency and responsiveness of a system using **vertical**, **horizontal** and **data access partitioning** techniques. Three major techniques for performance partitioning are :

2.2.2.4.1 Minimisation of the Interprocessor Communication Overhead

This overhead can be drastically reduced for a particular task by correctly partitioning processes and assigning the resulting sub-processes optimally to the various processors, to achieve a minimum communication between them.

A heuristic algorithm **PROXCUT** developed by C. J. Jenny and K. Haessig [40] was introduced for finding an approximation to the best placement of **computational objects**, such as **processes**, **data bases**, etc., in a distributed system.

In **PROXCUT**, a constraint graph is produced. Nodes of the graph are computational objects, edges (arcs) represent their constraint, and

the weights of the edges signify the **penalties** involved in not observing an assignment of objects prescribed in the constraint expression.

The graph can then be partitioned using a algorithm to achieve a optimum partitioning. **Exact** partitioning of a graph into a predetermined number of p components **(p-cut)** is considered to be an **np-complete problem**, where n equals the number of nodes. The number of required operations will not be bounded by any polynomial and hence will be useful for small graphs only. It follows that a **heuristic algorithm** must be found [41].

The algorithm is composed of two phases. A first phase generates an approximation by using a **maximum spanning-tree**, using Kruskal's algortihm [42], and a second phase improves on the approximation by using Ford/Fulkerson **maxflow/mincut** theory [43]. A full explanation of **maxflow/mincut** and the **spanning tree** can be found in appendix A. A third phase [44], was added to the **PROXCUT** algorithm to cater for the loading condition. It is claimed by the originators that the approximation for all practical purposes lies close enough to the optimum. Full details of this algorithm for the first two phases are given in [40].

2.2.2.4.2 Minimizing total cost of execution

The work at Brown University [41] by Stone and others closely resembles the above approach, but is restricted to two processors only. A set of weighted directed arrows is introduced, forming a **Module interconnection graph**. This weighted digraph corresponds closely to a constraint graph. The links between tasks represents the cost of communication and the link between one processor and a task represents the cost of running that task on the other processor in the

system.

In the case of a two processor system, the minimum weight cut **(mincut)** may be found very efficiently using a network flow algorithm [43]. This corresponds to an optimal assignment (i.e. minimises the sum of execution and running costs) indicated by the partitioning of the graph. This technique was adapted in [41] to an n-processor problem, where the solution is found by repeatedly applying the Ford-Fulkerson algorithm [43] to two out of n-processors. Stone's work based on static assignment was extended to find an optimal dynamic assignment of a modular program [45]. Two further costs were modelled onto the graph; these were the cost of dynamically reassigning from one processor to the other, and the cost of residence without execution. An example of static assignment will be given for illustrative purposes. **Fig.** 2.1 is shown with all the necessary information concerning the communication and execution costs.

The cut **(mincut)**, (obtained by applying Ford-Fulkerson algorithm), is shown by a thick line across the graph. The optimal assignment corresponds to a minimum weight **cutset**.

The total cost of execution for the static assignment of processes to the processor 1 in **Fig.** 2.1 is :-

Total running cost = Cost of (execution + communication)

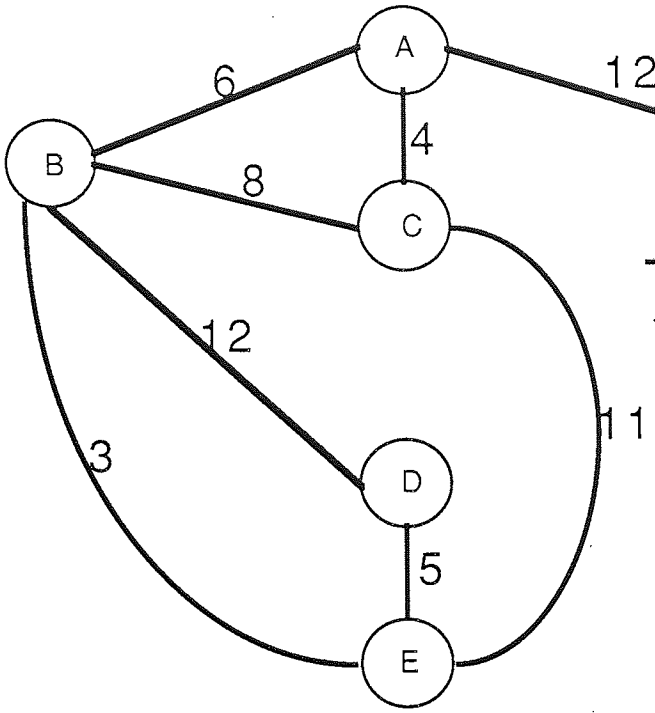
cost of execution = Module A + B + C + D + E

= 5 + 2 + 4 + 6 + 5 = 22

cost of comms. = Module A <----> Module F = 12

Therefore:

Total running cost = 34, and the total running cost for processor 2 is 16.



MODULE	P1 time	P2 time
A	5	10
B	2	∞
C	4	4
D	6	3
E	5	2
F	∞	4

Figure 2.1(a) Execution and Communication cost for distributed system

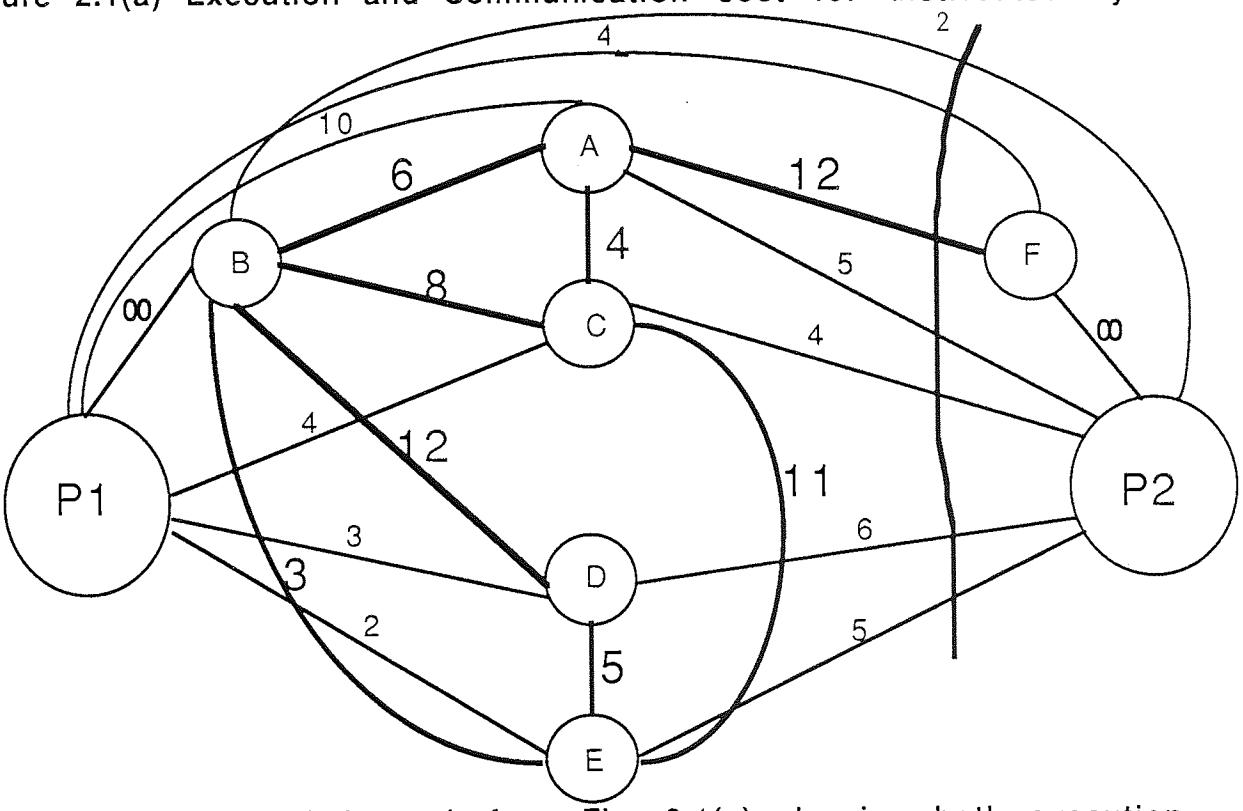


Figure 2.1(b) Modified graph from Fig. 2.1(a) showing both execution and communication cost on one graph

Figure 2.1

2.2.2.4.3 Extracting concurrency

A sequentially coded program can be executed more efficiently in a system that has more than one processor, if program segments which are amenable to parallel processing can be recognised [46]. The work of Ramamoorthy et al was concerned with finding an efficient scheduling mechanism, to minimize the scheduling overhead involved in real-time multi-processor execution of parallel processable segments of a sequential program [47].

This is achieved as follows. Given the **transition** graph of the system. An oriented graph in which the vertices (nodes) represent the single task and the oriented edges (directed branches) represent the permissible transition to the next task in the computational sequence. The graph is then represented in a matrix form, known as the **connectivity matrix**, C [48]. C is of dimension n x n (where n is the number of tasks) such that C_{ij} is a 1 if and only if there is a directed edge from node i to node j, and is 0 otherwise.

By analysis of the connectivity matrix, the **Maximally Strongly Connected (MSC)** subgraphs are determined (cyclic sub-systems, see section 2.2.3). A final reduced graph of the system can be derived by replacing each of the **MSC** subgraphs by a single node. Using the reduced graph a new connectivity matrix, T is obtained. Applying the **precedence partitioning** technique [46], i.e. provides an indication of the earliest time at which a task is initiated, the natural concurrent structure of the system is highlighted.

Note: the scheduling aspect obtained by the precedence partitioning technique is similar or identical to the technique used in management, known as **Critical path analysis** [49].

The **precedence partitons** can be obtained as follows. Using the connectivity matrix, T, a column (or columns) containing only zeros

is located. Let this column correspond to vertex v_1. Next delete from T both the column and the row corresponding to this vertex. The first precedence partition is $P_1 = \{v_1\}$. Using the remaining portion of T, locate vertices $\{v_{21}, v_{22}, ..\}$ which correspond to all columns containing only zeros. The second **precedence partition** P_2 thus contains these latter vertices. This implies that processes in set P_2 can be initiated and executed in parallel after the processes in the previous partition, P_1, have been completed. Next delete from T the columns and rows corresponding to vertices in P_2. This procedure is repeated to obtain precedence partitions $P_3, P_4, \ldots P_p$, until no more columns or rows remain in the T matrix .

The implication of this form of precedence partitioning is that if $P_1, P_2, \ldots, P_p$ correspond to times $t_1, t_2, \ldots, t_p$, the earliest time that a process in partition P_i, can be initiated is t_i. The duality of the procedure can provide an indication of the latest time at which a process may be initiated, by performing precedence partitions on the transpose of the C matrix. The timing obtained in this partition, serves as an input to the operating system to help in the scheduling of processes.

2.2.2.5 Assessment of goal-oriented partitioning

While each of the above goal-oriented approaches have merit in the development of software i.e. allowing the incorporation of resilience, concurrency, etc., it is not appropriate in attaining the goals elaborated in section 2.1 for designing the control structure of a decentralised system. This is because partitioning and assignment take place after the design effort (i.e using one of the software design

methods above) has taken place. However the software to be coded for the system can be parallelised and assigned optimally (goal ib and iii in section 2.2) to the distributed system using the partitioning technique to achieve the **performance** goal. The partitioning technique used to achieve the **reliability** goal could be used to make the system robust (resilient to failures). The partitioning concept of **change** and **growth** should always be considered when designing and installing a large and complex system.

2.2.3 Structural Partitioning

The design of any control strategy should reflect the structure of the continuous-time linear system being controlled such that it works in coordination and synchronisation with the physical properties of the system. The structural analysis can be used to make immediate and specific identification of any desirable modes of the physical system. Quantitative analysis alone presents problems for complex systems associated with rank determination solubility and design and also because initial data cannot be specified with sufficient accuracy [4].

It is assumed that some mathematical model of the linear system is available in which its properties are represented by a state space model. The infomation contained in a model of the system is based not only on input-output data but on dynamic properties (eigenvalues) as well, unlike the polynomial operator description of a system. In order to get insight into the dynamic properties of a physical system as shown in ref [50], the system can be mapped onto a directed graph (digraph) and decomposed algorithmically into so called **cyclic** and **acyclic** parts.

A linear system described by the state space equations:

$$\dot{X}(t) = AX(t) + BU(t)$$

$$Y(t) = CX(t)$$

where $X(t)$ is a vector of state variables of dimension N, U is a vector of control variables of dimension M, Y is a vector of output variables with dimension P, A, B and C are constant system, input and output matrices respectively.

This linear system can be transformed into a **directed graph (digraph)** which is closely related to conventional signal flow graphs. The **digraph** is derived from the A matrix by assuming that a directed branch exists from node j to node i for every $A_{ij} <> 0$ [51]. The states of the system are the nodes of the graph. The matrix B and C can be mapped into it in the same way.

The graph can be decomposed into **Cyclic** (**strong component**) and **Acyclic** components [51,52]. **Graph terminology** can be found in the appendix A. Certain graph theoretic results can be applied [5]. In particular, **the set of eigenvalues of a weighted digraph is the union of the sets of eigenvalues of its strong components.** The proof of this can be found in ref. [52]. Since the characteristic (eigenvalues) of a weighted digraph is completely determined by the characteristic of its strong component, nothing is known about how any individual strong component is related to any other [52].

The **acyclic** sub-system is associated with **zero eigenvalues**, and the **determinant** of every sub-system individually contributes directly to the determinant of the system as a whole. **As a consequence, it follows that the sensitivity of any particular eigenvalue to a perturbation in a system coefficent is zero for all coefficents not contained in the same sub-system as that eigenvalue.**

Any matrix is decomposable if an equivalent lower block

triangular form can be derived by a **similarity transformation** P^tAP in which P is a special permutation matrix and P^t its transpose; the derivation of this special matrix is described below. This results in a re-ordering of the associated digraph, so that the sub-systems of which the whole system is composed can be directly related to diagonal blocks of the transformation matrix P^tAP [51].

2.2.3.1 Decomposition

To achieve a **permutation matrix** [51], convert the system matrix A into its boolean counterpart by replacing each nonzero element by an **universal element** 1. Consider for example:

$$A = \begin{bmatrix} 0 & 0 & 0 & 2 \\ 1 & -3 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Therefore, the Boolean counterpart is as below,

$$A_b = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The graphical convention for this technique is that if an element of A_b, a_{ij} equals 1, there is a directed arc from node j to i, implying that the columns and rows of the matrix acts as source and sink nodes, respectively.

A **reachability matrix** can be constructed from the Boolean matrix A_b by observation of the digraph representation as shown in **fig. 2.2**. The elements of the **reachability matrix** R, $r_{ij} = 1$, if there is a path from node j to node i.

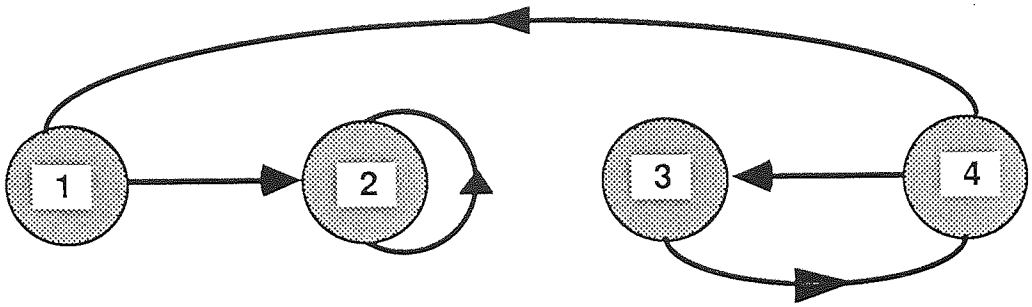


Figure 2.2 Graphical representation of a matrix

The reachability matrix for the above system matrix is as shown here,

$$R = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

However, this simplistic approach can become cumbersome if the system matrix is very large and therefore a systematic method is required to generate this matrix. It was shown in [52] to be:

$$R = A_b^0 \cup A_b^1 \ldots\ldots\ldots\ldots A_b^{n-2} \cup A_b^{n-1}$$

It is interesting to note that to achieve say A_b^2; it requires the boolean multiplication process [53]; in the **normal** matrix multiplication, an element of such a matrix would be derived as,

$$A_{ij} = \sum_{k=1}^{n} (A_{ik} \times A_{kj})$$

that is row-column **multiplication and add**, where as in the boolean case the elements are derived as row-column **logical and & or** i.e.

$$(A_b)_{ij} = \bigcup_{k=1}^{n} ((A_b)_{ik} \cap (A_b)_{kj})$$

It also becomes apparent that if one compares the element of the matrix A_b^2 with the digraph it appears that when there is a path of length 2 from node j to i, there is an entry at the element A_{ij} equal to 1. This fact is true for all powers up to and included N-1, where N is the

dimension of the state vector.

The final step in deriving the **permutation matrix**, requires one to express the boolean matrix, R, into **symmetric** and **antisymmetric** matrix form, as shown below. This means that the reachability matrix is the basis of decomposition of a system into cyclic and acyclic components.

In the boolean domain any matrix M_b can be expressed as the sum of a symmetric matrix

$$M_b \cap M_b^t$$

and an anti-symmetric matrix

$$M_b \cap \overline{M_b^t}$$

The derivation of these or proof can be found in the paper by Luce [53]. Thus continuing the example we have

$$\textbf{Symm} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad \textbf{Anti} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

sum of col. 1 1 2 2 sum of col. 1 0 2 2

By summing the columns as shown above, all columns with the same value in the symmetrical matrix are associated with the same **cyclic** component, however, each node in its own right is a **cyclic** component. But from the other matrix, one can observe that node 1 and 2 are separately identified as **acyclic**. From the second sum of columns vector, 1 0 2 2, one can generate the required **permutation matrix** P as follows [51]:

a) Scan the vector from left to right and find the position(s) of the column which has the same highest level, 3 and 4 would be found in this case.

b) Place it in another vector.

c) Repeat for the next highest level, 1 in this case, until a new vector of the same dimension is formed.

Hence we obtain a vector, 3 4 1 2, this is used in forming a permutation matrix by placing a universal element at column i; row (vector[i]), where i takes value from 1 to the dimension of the new vector.

The permutation matrix of the above is shown below,

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

This explicit decomposition of the system using the similarity transformation P^tAP is the basis for the **partitioning** technique to be used throughout this thesis (design objective 1). The technique allows the following design goals to be achieved:

2.2.3.2 Assessment of the structural technique

(i) When a linear system is partitioned into cyclic and acyclic sub-systems then the set of eigenvalues is the union of the sets of eigenvalues of its strong component (note that an acyclic sub-system is considered a strong component because each node in its own right is a cyclic component). Therefore, the partition will not **alter** the dynamic properties of the system. Hence, the design will be **fail safe** and **failure modes** can be overcome because each sub-system is stable even if any other sub-system becomes unstable (design objective 2).

(ii) The decomposed sub-systems can be mapped into the structure of control software. This extraction of **natural** concurrent processes is an **abstraction** idea, in which each process can be designed in isolation. This leads to a very modular design approach. The state vector (real-time database) of the linear system is partitioned

into disjoint subsets associated with each sub-system. Hence, necessary state information will need to be communicated between sub-systems in order to achieve decentralised control. Moreover, the analysis of the communication structure will help in the design of fault tolerant software (design objective 3).

(iii) The analysis of the computation and communication structure in chapter 5 will reveal that the concurrent software processes were partitioned into atomic actions and hence can produce a fail safe design (design objective 4).

2.3 COMMENTS AND CONCLUSION

2.3.1 Comments

Jensen et al raised several questions [36], which must be considered in the course of designing a distributed processing system. This chapter has considered in considerable length what partitioning criteria to use in designing a system. The question of when to partition in the design cycle and how long the designated assignment should persist needs some elaboration here.

Partitioning and assignment for a distributed processing system is a complex subject. To understand the issues involved Jensen et al [36] set up a model of the transformation from the computational requirement to the activity processed by a computing element. This transformation goes through several steps, and each step is performed by an **agent,** as shown in the **Fig.** 2.3.

The agent **programmer** will partition the system on the basis of intellectual manageability [27,38] and knowledge of the computational requirement, i.e. the specification. There are two extreme positions: First, the programmer is aware of the distributed nature of the real world processes or the designed implementation and will therefore, design explicit concurrent processes using a language like Occam [17]. Second, if the programmer has no knowledge of the distribution, then the **normal**, sequential, software design method will have to be utilised. In this case the next agent will have to extract those segments which can process in parallel and assign them to the appropiate destination in the implemented system.

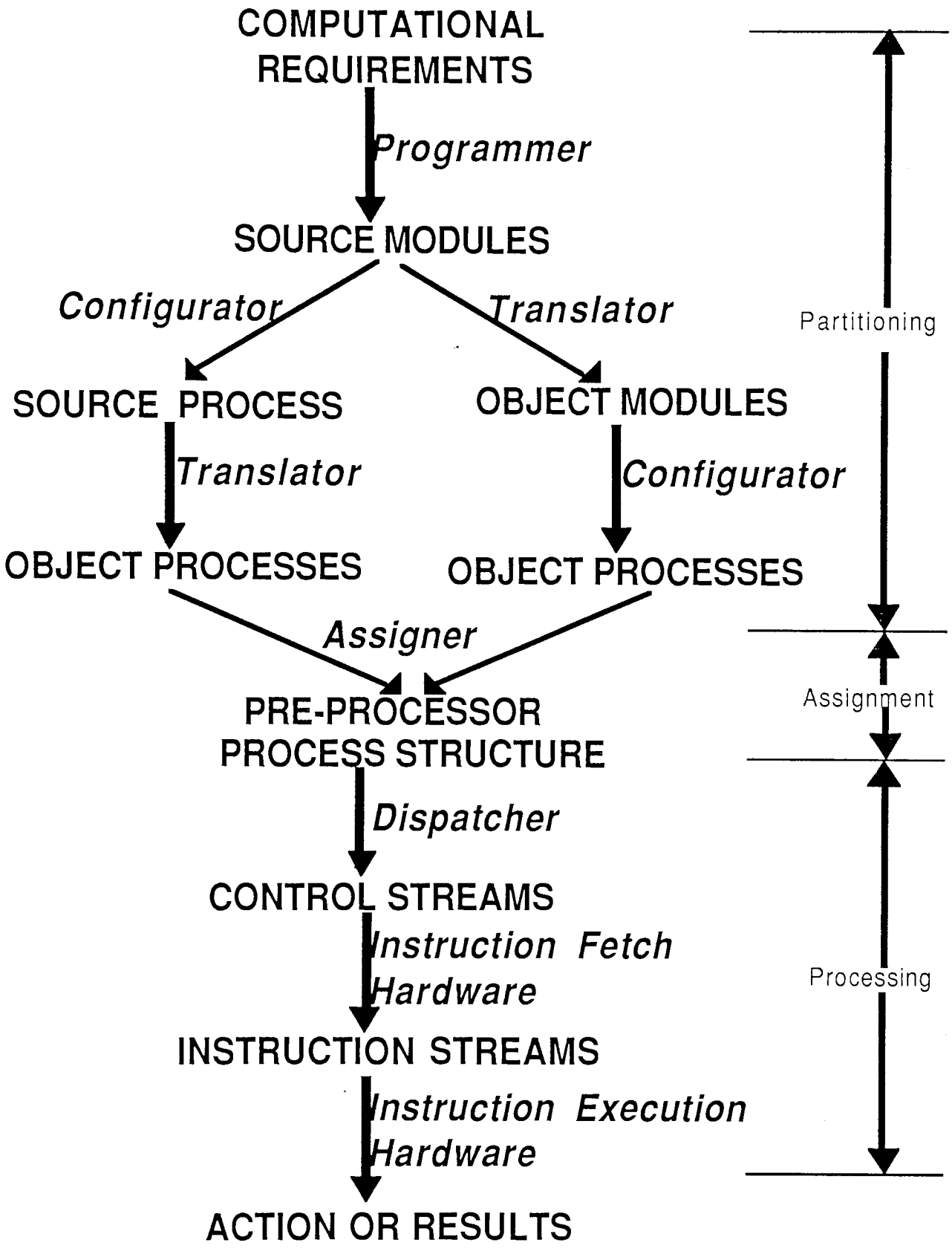


Figure 2.3 Transformation from computation to Activity

The **programmer** will dictate the characteristic of the **Translator**, the next agent in the design cycle. For example, in the second case of the above agent, there will be a requirement for a special type of compiler, which can recognise segments of the program which can execute in parallel [46]. The roles of the other agents in design transformation can be found in ref [36].

To answer question 3 posed in the introduction, the given assignment persists until a system or part of system ceases to perform to specification due to either component(s) failure or erroneous design. This will be discussed in greater detail in the fifth chapter.

2.3.2 Conclusions

The agent **System designer** was not explicitly mentioned in the discussion above. In the transformation process it should be the agent which partitions the system based on the physical structure of the system, as discussed in section 2.2.3, to extract the inherent parallelism of the processes in the system. These naturally identified boundaries, act as side walls to the conversation mechanism proposed by Randell [15]. This aspect will be dealt with in greater depth in the next chapter.

In this chapter it is shown that, the structural decomposition acts as a step-wise refinement of the requirement specification of the system; in other words it is a process of **Abstraction** at the highest level. The partitioning techniques such as Goal-oriented, Optimising performance, Extracting concurrency, etc., as discussed earlier in this chapter refine at a stage lower than those based on the physical structure. The most important conclusion drawn in section 2.2.3.1 was that a **structural partitioning technique** would need to be used as

an essential aim of this research if the design objectives identified in section 2.2.3.2 were to be achieved.

In general there is a need for a system design technique which encompasses as many of the stages in the software life cycle as possible to stop the rapid escalation towards a **system crisis**: integration and maintenance stages prove to be the most expensive in time and money to correct. Therefore, there is a need for an advance in system development techniques of similar magnitude to that represented by the introduction of the first high level languages.

The three techniques: **JSP**, **JSD** and **Object-Oriented programming** (Section 2.2.1) are systematic in their approach to the decomposition of a problem. The process of producing software, using any of the **JSD**, **JSP** or **Object-Oriented Programming** approaches, will tend to be convergent. Therefore, during integration and maintenance, this should prevent the introduction of errors due to misunderstanding other people's design. The use of these techniques will lead to the production of software systems that are modifiable, understandable and portable.

CHAPTER 3

3.0 CONTROL STRUCTURE OF A DISTRIBUTED SYSTEM

3.1 INTRODUCTION

In order to satisfy the four objectives listed in chapter 2, a literature survey was carried out. This chapter surveys literature only on the following three objectives: (i) systematic identification of the necessary communications for distributed control, (ii) to show that the partitions are the natural side walls to the conversation used in the design of fault tolerance software, (iii) to show that dynamic reconfiguration is possible. The literature survey concerning the estimation of state vectors in continuous and discrete-time systems is deferred until chapter 4, where the new approach to decentralised estimation is shown.

The physical distribution of the elements of a decentralised system will in general not be ideally matched to the information structures required to implement decentralised control over a particular physical system. The problem is then one of determining the conditions for controllability under a decentralised information structure and of using the communication links to implement the required system [6]. Therefore, a partitioning technique is required to match each controller to its corresponding physical sub-system and for the design of the minimal communication network.

The potential controllable sub-space is defined for each controller by its actuation units and the observable sub-space by its aquisition units. These can be determined by the technique developed in

Momen's thesis [4]. For any control station the controllable and observable sub-spaces may not be identical nor may they necessarily be unique to that control station for they may overlap similar sub-spaces belonging to other control stations. The use of a **structural partitioning technique** helps in matching the controllers to a physically decomposed system such that the sub-systems are decoupled dynamically. Therefore, the structural partitioning technique alleviates the problem of how to overlay a distributed control structure onto a distributed system.

Once the sub-space is allocated for each control station with the aid of the structural partitioning technique, the system structure under the decentralised control actions should be potentially controllable and observable. Further quantitative solutions should show these systems to be controllable and observable. In the general case it will be necessary to implement communications before the decentralised system can be made controllable and observable. The determination of the communications **necessary** for potential controllability and observability has been developed by Momen and Holding [6], but they ignored the effect of state variables, (due to other stations), which affect a given station. This information is required to estimate locally the state information of a given station because the database **(state vector)** of the controlled system is decomposed by the structural partitioning technique. Therefore, this thesis also emphasises the role of **communication** in the control of distributed processes. The essential theory of necessary communication is introduced in the next section and then used in the section concerning a new approach to the synthesis of a decentralised observer in a continuous-time linear system.

3.2 NECESSARY COMMUNICATIONS

In a distributed control system each individual controller will not normally have access to global system information about the initial state and will have insufficient information for the reconstruction of the state vector of the complete system. Dynamic systems of this type, where each of a number of control stations processes generally incomplete and nonidentical information about the state variables, are called decentralised systems. If the information fed back to each local input depends only on its corresponding subset of measurement variables the overall feedback strategy is called decentralised control [6].

To introduce the concept of Necessary Communications [6] in a Decentralised system we shall adapt several definitions from references 54, 55 and 56.

Definition 1

A structured matrix $\bar{A}$ is a matrix which has a number of fixed zeros at certain locations and arbitrary values elsewhere.

Definition 2

Two matrices A_1 and A_2 are structurally equivalent if there is a one-to-one correspondence between the locations of their zero and nonzero entries i.e. if the same structured matrix $\bar{A}$ corresponds to both A_1 and A_2.

Definition 3

If $\rho(A)$ denotes the rank of A, the generic rank $\bar{\rho}(A)$ is defined to be the maximal rank that $\bar{A}$ achieves as a function

of its nonzero entries.

3.2.1 Representation of system structure

It has been previously suggested in Reference 50 that any linear state space equation can be mapped onto a directed graph (digraph). When a system is mapped onto a digraph, the physical structure of the system will also appear in the digraph itself i.e the dynamic properties of the physical system. Therefore any visual inspection of this digraph will also reveal a realistic relationship between states, inputs, disturbances and outputs on the real process [57]. The use of such a graph has already been exploited in studies of controllability and observability for large-scale systems [51].

First, we give a short summary for these results for the conventional linear system which appears in references 51 and 57.

$$\dot{X}(t) = AX(t) + BU(t)$$
$$Y(t) = CX(t) \qquad \text{---------- (3.1)}$$

where $X(t)$ is a vector of state variables of dimension N, U is a vector of control variables of dimension M, Y is a vector of output variables with dimension P.

The Boolean form of a matrix A_b is generated from A by substituting every nonzero entry with unity. The digraph of the system can be derived from the system Boolean matrix A_b. For example, consider the Boolean system matrices.

$$A_b = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad B_b = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad C_b = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

The digraph of the system is shown in **Figure** 3.1. The digraph is constructed from the Boolean matrix A_b, as follows if an element of

A_b, a_{ij} equals 1, than there is a directed arc from node j to i.

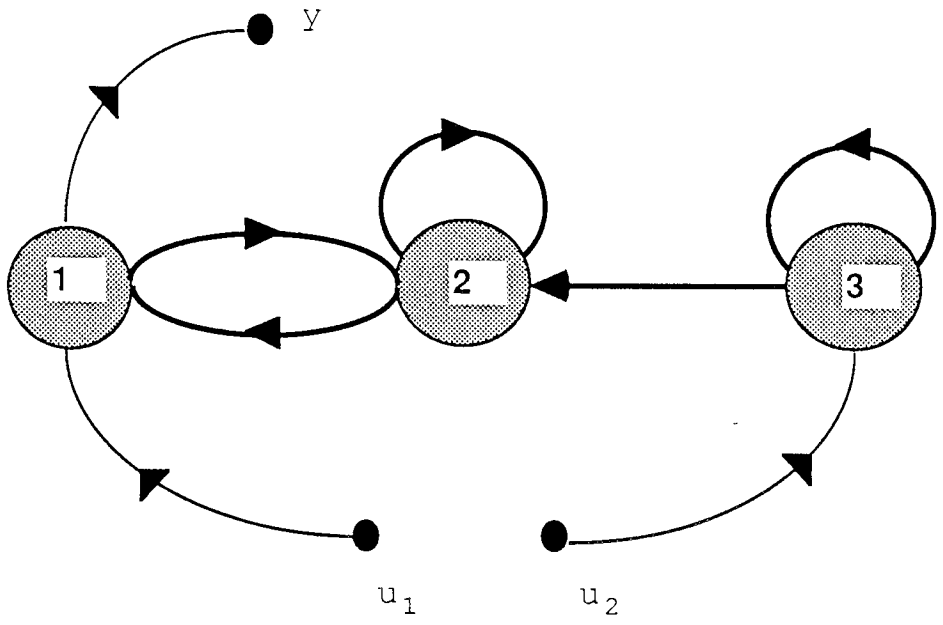


Figure 3.1 Digraph of simple system

An alternative matrix description of a digraph is that given by its **reachability matrix** R and for the above example in **Fig. 3.1**, we could write

$$R = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

in which $r_{ij} = 1$ if the i^{th} node has a directed path from the j^{th} node, and $r_{ij} = 0$ otherwise. The reachability matrix can be used in the analysis of digraph properties but it is not by itself sufficient for a complete analysis of the dynamics of a system as the so called **term-rank** properties must be considered in all situations [51]. The term-rank of the Boolean matrix A_b is equivalent to the generic rank of the matrix A, i.e.

$$TR[A_b] = \overline{\rho[A]}$$

Definition 4

The term-rank of any Boolean matrix A_b is the number of elements which exist in the maximal permutation matrix contained in A_b. Such a maximal permutation matrix is not neccessarily unique and this fact has considerable significance in the structural analysis of decentralised systems, this will be illustrated later.

For the general system given in **eqn.** 3.1 the well known **Kalman** criteria are

a) For controllability,

$$\text{Rank } [B,\ A^1B,\ A^2B, \ldots\ldots A^{n-1}B] = N$$

b) For observability,

$$\text{Rank} \begin{bmatrix} C \\ CA^1 \\ CA^2 \\ \cdot \\ \cdot \\ CA^{n-1} \end{bmatrix} = N$$

From a structural point of view it has been shown that the single numerical criterion for controllability, i.e. the above Kalman criteria, can be replaced by two separate and necessary conditions. The same applies to observability by duality [51].

(a) A term-rank condition

$$TR[A_b,\ B_b] = N$$

(b) A reachability condition that all states are reachable in a digraph sense from at least one input. That is if there are no nonzero rows in the Boolean matrix defined by $R\ V.\Lambda\ B_b$. Where R is the reachability matrix of A_b and B_b is the boolean equivalentof B.

The boolean matrix multiplication is represented explicitly by the operation $\vee.\wedge$ borrowed from **APL** notation [58].

Associated with these will be the concepts of structural (or potential) controllability and observability. A system (A, B) is structurally controllable if there exists a system structurally equivalent to (A, B) which is controllable in the Kalman sense [56].

3.2.2 Controllable and Observable sub-spaces of a general control station in a Decentralised System.

A decentralised system is said to be controllable under a decentralised information structure if there exists a decentralised control law which transfers any initial state, which is not known to all the control stations, to the origin in a finite time interval [59]. It is possible that a decentralised system may not be controllable and observable under the decentralised information structure using only the local information sets of the control stations; in this case communications may be required to enhance the local information sets to achieve controllability. The problem of controllability and the role of communications are considered in detail in the following sections.

Consider the decentralised control using s control stations of the complete linear continuous-time system described by the equation

$$\dot{X}(t) = A\,X(t) + \sum_{i=1}^{s} B_i\,U_i(t)$$

$$Y_i(t) = C_i X(t) \qquad (i = 1, 2 \ldots s)$$

where $X(t)$: vector of state variables, dimension N

$U_i(t)$: vector of control variables, control station (i), dimension m_i

$Y_i(t)$: vector of output (observation) variables, control

station (i), dimension p_i

A, B_i, C_i : constant system, input and output matrices

The system is informationally decentralised since, in general, the control stations have different observation vectors.

Each control station in a decentralised system controls a sub-space of the state variables of the system. Similarly, each station observes a sub-space of the state variables of the system. For any control station the observable sub-space may not be identical to its controllable sub-space, nor may they be unique to that station for they may overlap similar sub-spaces belonging to the other control stations.

We assume that the system is jointly controllable and jointly observable, i.e.

$$TR[A_b,\ B_b] = N$$

and

$$TR\begin{bmatrix} A_b \\ C_b \end{bmatrix} = N$$

and that all the state variables are reachable from at least one input and that each state can reach at least one output. This assumption does not necessarily mean that the system is controllable or observable from a single control station. To simplify the notation, we shall now consider that all the matrices and vectors are in a Boolean form, for example, $A = A_b$.

It is well known that the controllable sub-space pair (A,B) are not generally unique [60]. This means that a station i of a decentralised system may have more than one controllable and observable sub-space. To determine these sub-spaces for every control station $1,2,i,\ldots,s$ in the decentralised system we shall follow the procedure in references [4,6,54].

Step 1: the reachable sub-space from U_i is a subset of all

state variables which can be reached from any input of control station i. It can be expressed as a column vector as follows

$$R_{Ui} = V/[R\ V.\Lambda\ B_i]$$

where R is the reachability matrix for the whole system, and the notation V/ means logical summation over the columns.

The ones in the vector R_{Ui} represents the state variables which are reachable from control station i.

Step 2: the reachable sub-space to Y_i, the set of inputs of control station i, spans the set of all state variables from which Y_i can be reached. It can be determined as a row vector as

$$R_{Yi} = V/[C_i\ V.\Lambda\ R]$$

the notation above means logical summation over the rows

Step 3: since we can deal with the pair (A_{ui}, B_i) as a separate sub-system which is reachable from U_i, then to determine whether the sub-system is controllable or not, we must test the term-rank of the matrix

$$[A_{Ui},\ B_i]$$

in which the N x N matrix A_{Ui} represents the reachable sub-system of the input U_i. This matrix is formed from the system matrix A (A_b) by equating all the elements in the i^{th} row and column to zero if the i^{th} state is not reachable from U_i.

Generally, we have two possible conditions:

(a) $TR[A_{Ui},\ B_i] \geq r_{ui}$

(b) $TR[A_{Ui},\ B_i] < r_{ui}$

where r_{ui} is the number of ones in the reachability vector R_{Ui}, i.e. the number of states reachable from control station i.

It is clear that if condition (a) is satisfied, then the sub-system A_{Ui} is potentially controllable from control station i, and the controllable sub-space, which is unique in this case, will be

$$K_i = R_{Ui}$$

However, if condition (b) is satisfied, there are one or more of the state variables which are reachable, but not controllable. The number of state variables that are controllable from U_i is equal to

$$TR[A_{Ui},\ B_i]$$

The term-rank of the matrix $[A_{Ui},\ B_i]$ can be determined from any of the valid entries in the permutation matrix. This means generally, that the control station i may have more than one controllable sub-space. All the possible choices which satisfy the $TR[A_{Ui},\ B_i]$ will be represented by the set of matrices

$$^{j}[A_{Ui},\ B_i] \quad j= 1,2\ \ldots$$

Step 4: we can determine the potential observable space of station i, i.e. by using the dual technique of above (steps 1-3) which determine the potential controllable sub-space, by testing the term-rank of the matrix

$$\begin{bmatrix} A_{Yi} \\ C_i \end{bmatrix}$$

The result of this test is one of two possible conditions

(a) $$TR \begin{bmatrix} A_{Yi} \\ C_i \end{bmatrix} \geq r_{yi}$$

(b) $$TR \begin{bmatrix} A_{Yi} \\ C_i \end{bmatrix} < r_{yi}$$

where r_{yi} is the number of ones in the reachability vector R_{Yi}, i.e. the number of the state variables that reach any of the outputs of control

station i.

If condition (a) is satisfied then the potential observable space, which is unique in this case, will be

$$M_i = R_{yi}$$

However, if condition (b) is satisfied then control station i does not have the potential to observe R_{yi}. The number of states which are observable is given by

$$TR \begin{bmatrix} A_{Yi} \\ C_i \end{bmatrix}$$

The term-rank of the matrix $[A_{Yi},\ C_i]$ can be determined from any valid entries in the permutation matrix. This means generally, that control station i may have more than one observable sub-space. All the possible choices which satisfy the $TR[A_{Yi}, C_i]$ will be represented by the set of matrices

$$^j[A_{Yi},\ C_i] \quad j= 1,2 \ ...$$

Step 5: A control station may possess controllable sub-spaces that cannot be used to achieve, even with another control station, full controllability of the system. This kind of controllable sub-space is known as an **incompatible controllable sub-space**.

In the decentralised system, the compatible sets of controllable sub-spaces of the control stations are sets which satisfy the condition

$$^jK_1 \ V \ ^jK_2 \ V \ ..... \ ^jK_n = \text{unit vector}$$

Step 6: A duality exists for the observable sub-spaces. Therefore, the compatible sets of observable sub-spaces of the control stations are sets which satisfy the condition

$$^jM^t_1 \ V \ ^jM^t_2 \ V \ ..... \ ^jM^t_n = \text{unit vector}$$

NOTE: Here we slightly deviate from Momen's later work (his thesis) which uses implicit feedback due to other controllers [4],

because this thesis is concerned with resilient control structure and therefore, the work of Momen and Holding will be followed [6]. Appendix B details these additional steps, for enlarging the above sub-spaces. The reason for this departure is that if any control station fails or malfunctions the potential controllable and observable space of the other station or stations will be affected if its space was derived using implicit feedback signals. This error in space derivation will propagate to the other stations because its enlarged sub-space was dependent on the one previously affected and the whole controllable structure will collapse.

Step 7: The common sub-space is found for each of the stations

$$L_i = K_i \cap M^t{}_i$$

Step 8: In decentralised structure the union of all the common sub-spaces must be the unit vector, i.e. the following equation must hold if the system is to be controllable under the decentralised information structure

$$\bigcup_{i=1}^{s} L_i = \text{unit vector}$$

However if the above condition does not hold then in such a system necessary communications are the minimal communication required to control the complete system. The necessary communications for the system can be determined by considering the disjoint sub-space which can only be controlled by communications as derived in reference [6].

The necessary communication, for station i

$$I_i = [K_i \cap \overline{M_i^t}] \cap [\overline{L_1} \cup \ldots\ldots \cup \overline{L_j}]$$

for $j=1,2, \ldots n;\ i \neq j$

3.3 DISTRIBUTED CONTROL SOFTWARE

The problem we are faced with now is how to design software for a decentralised system which should be robust (resilient to failures). The structural partitioning technique decomposes a physical model of a linear system into cyclic and acyclic sub-systems. Each sub-system controller's software will comprise (i) local estimation of its partitioned state vector, (ii) a control function and (iii) a communication interface to other controllers. In general, if a sensor and an actuator is allocated to each sub-system and then, as a direct result of the structural partitioning of the physical model, two communicating concurrent processes can be **extracted**: one for estimation and the other for the control function.

The design of information processing for distributed control software derived from the above approach leads to processes which are robust. This is because the software processes are mapped from linear sub-systems which are dynamically decoupled. It may be necesssary to consider the software fault tolerant measures applicable to such a loosely coupled distributed system to make it more robust. Moreover, analysis of the communications between the processes controlling the linear system leads to a method for designing fault tolerant software [61]. A summary of existing and recent software fault tolerance techniques is given in the next section. This framework is required to show that the software processes derived by the structural partitioning technique are side walls to the conversation [15] (objective (ii) in section 3.1).

3.3.1 Software fault tolerance

Due to the complexity of computer systems, it is generally impossible to obtain a system which is completely free from faults [62]. System malfunctions can be caused either by hardware or software faults.

To develop a reliable software system a wide range of techniques has to be applied to all stages of the software life cycle. Methods are being developed for increasing the correctness of design [31] by using correctness proofs. Fault avoidance methods can be used in the design as summarised in reference [12]. System and program design methods, like **JSD**, **Object Oriented** approach and **JSP** will eliminate some of the design faults by imposing a software engineering discipline on the design. Nevertheless, faults are likely to remain.

The basic scheme in fault tolerant systems is to **neutralise automatically** the effects of a fault with **protective redundancy.** The primary aim of protective redundancy in hardware is to overcome the wear and tear of physical components. Hardware fault tolerance has been studied for a long time [63] and, with the increase in hardware reliablity due to advances in technology, hardware structures have been developed which will cope, with a high degree of probability, with these faults.

The primary aim of protective redundancy in software is not to overcome the wear and tear problem because software does not age. The problem is as, Randell [15] states, that all software failures result from **design faults.** In software fault tolerance the aim is to overcome the design fault. The complexity in software design can be several orders of magnitude greater than that of the hardware design due to the number of possible states of a software system [61]. Therefore we cannot test all states in a finite time so faults are likely

to remain undetected and it is necessary to cope with them [20].

The faults can be classified into two categories: **anticipated** faults and **unanticipated** faults. The examples of the former category are division by zero, floating point overflow, etc. The latter suggests by its name that faults are very unusual, for example, intermittent hardware faults, a 'bug' in the program, etc.

The structures to enforce fault tolerance must perform the following tasks when a fault occurs:

(1) detect that a fault has occurred

(2) assess the extent of the damage that has been caused

(3) repair the damage

(4) treat the cause of the fault

Once the error detection has taken place, the damage repair must take place. There are two strategies for repairing the damage. These are classified as **Forward** and **Backward** error recovery techniques [64].

Forward error recovery techniques are generally used for recovering from anticipated faults and backward error recovery for recovering from unanticipated faults [65].

3.3.2 Recovery Block

The recovery block mechanism [15,66] provides a backward error recovery scheme for conventional sequential systems. The technique partitions the software into recovery blocks. On entry the current state of the system is saved. (The assumption is that this is a correct state of the system). If a fault is detected, the state of system is restored. The system now continues using some alternative course of action, so as to avoid the original problem. It uses a similar mechanism to the **stand by spares** approach used in hardware systems [67].

The recovery block scheme is described by the syntax given in fig. 3.2

```
ENSURE      ACCEPTANCE TEST
BY          PRIMARY PROCESS
ELSE BY     ALTERNATIVE PROCESS
ELSE BY     ALTERNATIVE PROCESS
ELSE        ERROR PROCESS
```

Figure 3.2 Recovery Block Outline

On entry to the recovery block a **recovery point** is established and the primary block is entered. On completion of the primary block the acceptance test is executed. If the test does not raise an exception the recovery block is exited. However, if an exception is raised the recovery point is restored, the next alternate block is executed and the above procedure is repeated.

Recovery blocks can be used in concurrent systems, but some form of recovery coordination is required for correct operation of this recovery mechanism. If recovery points of interacting processes are not properly coordinated, then an intolerably long sequence of rollback propagations, called the **domino effect** [15], can occur.

The domino effect can happen when two particular circumstances occur:

1) The recovery block structures of the various processes are uncoordinated, and take no account of process interdependencies caused by their interactions.

2) the processes are symmetrical with respect to failure propagation - either member of any pair of interacting processes can cause the other to back up.

3.3.3 Conversation Block

An abstract construct termed a conversation was proposed in 1974 [15] as an aid to the structuring of properly coordinated error detection and backward error recovery in interacting processes. A conversation [68,69] is an extension of the recovery block technique, to two dimensions. Like recovery blocks, conversations provide a wall which serves to limit the damage caused to a system by errors. The boundary of a conversation consists of a recovery line, a test line and two side walls. A recovery line is a coordinated set of the recovery points of interacting processes that are established before interactions begin. A test line is a correlated set of the acceptance tests of the interacting processes. A conversation is succesful only if all the interacting processes pass their respective acceptance tests. If any acceptance test is failed, all the processes must roll back to the recovery line and retry their alternate blocks.

The basic program structuring rules of the conversation scheme can be summarised as follows [69]:

i) A conversation defines a recovery line as a line which processes in rollback cannot cross.

ii) Processes enter a conversation asynchronously.

iii) A conversation contains one or more interacting set of processes aiming at the same or similar computational results.

iv) A conversation defines a test line which is an acceptability criterion against which the results of an interacting set of processes are assessed. A test line can thus be viewed as a single global acceptance test called a conversation acceptance test.

v) Processes cooperate in error detection, regardless of the source of the error.

vi) Two sidewalls defined by a conversation imply that the process participating in the conversation must neither obtain information from, nor leak information to, a process not participating in the conversation. That is, no information smuggling by processes in a conversation is permitted.

One aspect of the thesis is to design a system with resilient control structures for use in real-time control. The design method using the backward error technique by the placement method (Tyrrell [16]) for asynchronous concurrent processes will be utilised.

Note: the above ideas (sect. 3.2-3.3) will be used to demonstrate that the structural partitioning technique used in decomposing a linear system not only eases the sensor/observer (estimation) and actuator/control problem (chapter 4), it will also ease the construction of a fault tolerant system because of the disjoint values of the partitions of the state of the physical system, and the overlay of software processes necessary to control such a system (chapter 5).

The final section below is introduced to show that dynamic reconfiguration of software processes associated with sensors or actuators, which has been ignored for real-time control systems, is possible (objective iii in section 3.1).

3.4 DYNAMIC RECONFIGURATION

The flexibility provided by a distributed environment can be exploited by a variety of techniques, ranging through elaborate dynamic reconfiguration, redundant processing, to fault isolation and explicitly considering fault tolerance of software [70]. It was recognised by Boebert et al [70], that it would be catastrophic to ignore the need to provide fault tolerance for real-time control applications. However, the case for providing support for dynamic reconfiguration was rejected because, the processors in distributed real-time control systems are typically located near the sensors and actuators they serve. They argued that reconfiguration is rendered ineffective by the inability to move the function of the external devices.

There are few reports in the literature in the area of multiprocessor systems dedicated to a particular real-time task, such as control of industrial processes [71]. They all argue, even to the present, against the possibility of dynamic reconfiguration of processes serving sensors and actuators.

The migration between other **computing elements** of software components which are not specific to actuators or sensor control can be used to enhance performance by load sharing in time and space and to enable software components to tolerate hardware failures. Some reconfigurable architectures are surveyed in [72] with the aim of evolving a superior version. The work at Brown University [41,45] shows how the load can be shared in time and space using a graphical modelling technique. (See also chapter 2).

Some of the related work concerning dynamic reconfiguration of a **System** can be found in [33,73,74,75]. Static approaches exist like the **Polyproc** system [76]. However, static configuration is not applicable to the work presented here. Dynamic reconfiguration of a

system is necessary because a large computer system should be introduced into the environment gradually. This will ensure that as a system is introduced it will be integrated and work in a safe and coherent manner. The ability to modify and extend a system while it is running, will accommodate unpredictable changes at unpredictable times.

Two of the relevant approaches to configuration of a **System** dynamically will be summarised here. First **MASCOT** [33] and then **Conic** [73].

(i) **MASCOT** [33] provides a framework for constructing systems for real-time applications on a single computer. The **MASCOT** kernel provides scheduling and synchronisation primitives. The processes communicate through **IDAs** using access procedures. The **MASCOT** commmand interpreter allows a system to be configured dynamically by the command **Form**. This creates a set of activities (processes) from their root procedures and interconnects these acitivities by substituting the addresses of the **IDAs** for the corresponding formal parameters and the root procedures. Each **Form** command creates a named sub-system, which can be deleted subsequently by a **Remove** command. The availability of component connection with component instantiation means that reconfiguring connections can be done but only by deletion and re-instantiation of components.

(ii) **Conic** [73,77] provides on-line dynamic configuration: creation, deletion and interconnection of software components. Facility for system building and dynamic configuration are provided by keeping configuration separate from the programming language. The concepts of a module and message primitives have been added to the programming language Pascal. **Conic** is then translated to standard Pascal for execution. The extra functionality of tasking and message

communication is incoporated as procedure calls to the kernel. **Conic** provides a configuration language to specify system configuration. The system configuration specification identifies the module type from which the system is constructed, declares the instances of these types which will exist in the system and describes the interconnection of instances. These three functions are provided by the constructs:

a) **Use**: the construct provides a context of module type from which a system is constructed.

b) **Create**: this construct declares named instances of module types which will exist in the system.

c) **Link**: this interconnects modules, by binding the **entrypoint** (input port) of a module to the **exitport** (output port) of a second module instance ie the intermodule communication channels.

To support dynamic reconfiguration three functions, **Remove**, **Delete** and **Unlink** are provided to perform the inverse function to **Use**, **Create** and **Link**. Dynamic configuration is supported by the **Configuration Manager**, which translates a request to change the system, expressed in the **Conic** configuration language, into commands to the distributed operating system to execute re-configuration operations.

3.5 COMMENTS and CONCLUSIONS

Particular emphasis was given to the role of communication in the control of distributed processes, and to the partitioning technique necessary to support the design of distributed and decentralised control system with resilient structures.

Chapter 2 surveyed the existing partitioning techniques and adopted the structural partitioning technique. This technique allows one to answer the fundamental question posed in section 2.1, and the method with **communication** allows one to satisfy several goals: three were dealt with in this chapter (section 3.1) and are summarised below:-

(i) Identification of necessary communication

(ii) Fault tolerant software

(iii) Dynamic reconfiguration.

3.5.1 Necessary Communications

The essential theory of necessary communication was introduced. It was pointed out that the existing work on identifying the necessary communication to control the system in a decentralised fashion [6], ignored the effect of state variables, (due to other stations), that affects a given station. An extension of this work is shown in chapter 4. This extension identifies those state variables of other stations which interact with a given station.

3.5.2 Fault Tolerant Software

This section surveys the existing and most recent technique used for designing fault tolerant software for real-time systems. The

boundaries derived using the partitioning technique developed in chapter 2 will be to shown to be the side walls of the conversation in chapter 5. In other words, the processes mapped from the decomposed linear systems are atomic.

This statement has two implications. (i) A recovery block is required for each partition at the outermost boundary level and conversations are required for the sub-processes within the partitioned boundary. (ii) the conversation is identified **before coding** unlike other techniques [16]. The partitions identify the fault tolerant mechanism at an abstract level.

3.5.3 Dynamic Reconfiguration

The literature survey carried out in this section shows that the dynamic reconfiguration of processes has been ignored in the past, especially when those processes are servicing the sensors or actuators of the system being controlled. In chapter 5, it will be shown, by example, that the technique of dynamic reconfiguration can be used with these types of processes in real-time control systems for certain classes of systems.

Note: the work on the dynamic configuration of a **system** will not be addressed in this thesis. However, there is a real need for large embedded computer systems to accommodate evolutionary change, particularly those systems with expected long-life time. They need to evolve as human need changes, technology changes, and the application environment changes.

CHAPTER 4

4.0 OBSERVATION, ESTIMATION AND COMMUNICATION STRUCTURE

4.1 INTRODUCTION

In the last decade there has been a great deal of interest in the decentralised control and estimation of the state vectors of large complex systems [78]. This chapter deals mainly with the decentralised estimation of the state vector of a large complex system in continuous and discrete-time using the **Luenberger observer** technique [79] and the **Kalman Filter** technique [80,81] respectively.

The **Luenberger observer** technique [79] is a well known fundamental technique for the estimation of the states of a centralised system. It was shown by Siljak and Vukcevic [8] also to have application for decentralised systems. Siljak and Vukcevic synthesised a decentralised observer from local measurements, using output decentralisation, within an interconnected sub-system constraint. Their state space representation of the sub-system equations must be defined as shown below:

$$\dot{X}_i = A_i X_i + \left(\sum_{\substack{j=1 \\ j \neq i}}^{s} A_{ij} X_j\right) + B_i U_i$$

$$Y_i = C_i X_i \qquad i = 1,2,\ldots,s$$

Where,

$$A_i = \begin{bmatrix} 0 & 0 & \dots & 0 & -a1 \\ 1 & 0 & \dots & 0 & -a2 \\ & & \dots\dots\dots & & \\ 0 & 0 & \dots & 1 & -an \end{bmatrix} \qquad C_i = \begin{bmatrix} 0 \\ 0 \\ . \\ 1 \end{bmatrix}$$

and the interconnection matrix $A_{ij} = (a_{pq}{}^{ij})$ such that

$$a_{pq}{}^{ij} = 0, \quad p > q$$

where $p = 1,2,\dots,n_i$ and $q = 1,2,\dots,n_j$

Note: to simplify the notation all square matrices which are represented by A_{xx} will be denoted by A_x throughout this chapter and the thesis.

However, even though the sub-observers are made individually stable, the composite observer constituted by the individual sub-observers must be checked for stability by the **vector Liapunov function** concept. The Liapunov function is a well known technique for checking stability of a system without actually calculating its eigenvalues.

Recently, several researchers have demonstrated techniques for estimating the state vectors in a decentralised fashion [9,10,82]. However, the concept underlying all their work stems from Siljak and Vukcevic's approach.

First, the work of Geromel and Yamakami [9] extends the work of Siljak and Vukcevic to continuous-time linear systems. Their method tries to overcome the unrealistically large feedback gains and difficult numerical procedures experienced by other workers and is demonstrated in **ref. 9**. The technique still requires the **vector Liapunov function** for checking the stability of the complete system within a different interconnection sub-system structure constraint. They also present a solution for the discrete-time case, which they claim to be the first ever presented. Their state space representation

of the sub-system is defined as in Siljak et al above and the interconnection matrix is defined such that any two sub-systems i and j must be factored as

$$A_{ij} = B_i L_{ij} C_j, \quad i \neq j=1,2,\ldots,s$$

so that the overall continuous system is stable and the decentralised feedback gains are realistic [9].

Second, the work of Viswanadham and Ramakrishan [10] reformulated the decentralised problem for interconnected systems into that of synthesising an observer assuming that no information transfer is possible. This is termed the **unknown input observer** problem.

A common problem with the above approaches concerns the stability of each sub-observer. This is because the stability of the observers of the interconnected sub-system has to be checked using the **vector Liapunov function** in all of the above approaches [8,9,10]. This is because the interconnected sub-systems are not dynamically decoupled. Therefore, other sub-observers will become unstable due to the failure of any one of the individual sub-observers. Another problem is that a local estimate requires the transmission of any necessary estimated state information from the appropriate sub-observers. Hence, the reliability of the sub-observers cannot be guaranteed if any of the sub-observers fails for any reason. These problems were pointed out by Khun and Schmidt [82].

This chapter shows firstly a **new technique** for decentralised observation which **circumvents the stability problem** highlighted above in the continuous-time case by using the **cyclic** structural properties for the first time in the design of the decentralised observers. Secondly, this chapter **identifies systematically** the **necessary state information** which a sub-observer requires from the other sub-observers to estimate locally. Thirdly, this chapter

shows in the continuous-time case how **fault tolerant sub-observers** may be constructed. Finally, the idea of decentralised estimation of the state vector is carried forward to the discrete-time system using a **Kalman Filter**; this is a technique for estimating the states for a discrete-time system when the presence of noise in the system is taken into consideration. This presents **original work** which shows how **decentralised estimation can be achieved by the Kalman Filter**.

This chapter is split into two major sections. The first section is for the continuous-time and the other is for the discrete-time case. The continuous-time section shows an application of the decentralised observation process (based on the decentralised control technique of Evans and Schiza [7,83]) which circumvents the need for stability analysis of the complete interconnected sub-sytems (as done by the other researchers [8,9,10]) provided the **natural** (cyclic) boundaries of the system are adhered to (see chapter 2). Hence, it is a simpler numerical process.

The aforementioned publications [8,10] demonstrate that the separation property holds for their decentralised estimator; this property also applies for the decentralised technique to be presented here, and is demonstrated in this first section.

The continuous-time sub-section derives systematically the necessary communication required by decentralised controllers for observation processes. This augments the work of Momen and Holding [6] who identified the necessary communication required to make the system controllable in a decentralised scheme (see chapter 3 for fuller details). This form of communication is known as **secondary communication** to distinguish it from the work of Momen et al [6].

In this last sub-section, first part, certain matrix criteria are also presented which lead to robust decentralised observers by

eliminating communication between observers.

The discrete-time section shows the **Kalman Filter** has an application for estimating the states of the system in a decentralised fashion, if again the **natural** boundaries of the system are adhered to. This approach shows a considerable **improvement in computation time**, because the **order** of the estimation problem is reduced. That is, instead of using one complex Kalman filter method for estimation, this method uses **several** small order **asynchronous Kalman filters**. Researchers at the Queen's university, Belfast are trying to speed up the **Kalman filter**, by redesigning the standard algorithm to use **systolic arrays** [84,85].

4.2 CONTINUOUS-TIME SYSTEM

4.2.1 Decentralised Observers

4.2.1.1 Introduction

Although estimation and control problems in decentralised systems are of equal importance [8], decentralised control has received much wider attention than its estimation counterpart. This difference in popularity is due in part to the need for effective decentralised control schemes in constructing decentralised observers for state estimations. In this section a decentralised observer for a continuous linear system will be illustrated using the decentralised control scheme of Evans et al [7].

4.2.1.2 Decentralised Control

Results about the decentralised control problem achieved by researchers for continuous linear systems have appeared in the literature [8,9,78]. These methods suffer from deficiencies; they may have a specific structure constraint, or limited practical applications, or provide the designer with an unrealistically large gain, or involve difficult numerical processing.

The work of Evans et al has shown that decentralised control can be achieved if one stays within a **natural structure** constraint that is within a **cyclic** sub-system. The properties of **cyclic** sub-systems are explained earlier in this thesis in Chapter 2. These sub-systems (or systems) achieved by structural partitioning suggest that the sub-system cannot be further decomposed using structural attributes. However, quantitative analysis of these sub-systems, using

the **sensitivity matrix** [7], shows that further decomposition may be possible.

4.2.1.3 Decentralised Observer

This section shows a new way of achieving decentralised observation by applying the decentralised control technique of Evans and Schiza [7].

The Decentralised Observer equation can be derived by considering the **one-shot** observation equation as described in ref [80]. Consider a linear system described by **equation** 4.1

$$\dot{X}(t) = AX(t) + BU(t)$$

$$Y(t) = CX(t) \qquad \text{----------(4.1)}$$

where,

$X(t) \in \Re^N$

$U(t) \in \Re^M$

$Y(t) \in \Re^P$

A, B and C are constant N x N, N x M and P x N matrices respectively.

It is assumed that all the matrices and vectors except $\dot{X}(t)$ and X(t) in **equ.** 4.1 are known. The only thing preventing a direct reconstruction of the system in order to arrive at the state vector itself is the lack of knowledge of initial conditions. If an estimate of the state vector, $\hat{X}(t)$, is obtained, a measure of error between the plant and the estimator model output, E(t), can be determined i.e.:-

$$E(t) = Y(t) - C\hat{X}(t) \qquad \text{----------(4.2)}$$

This error vector can be used to improve the estimation

$$\dot{\hat{X}}(t) = A\hat{X}(t) + BU(t) + GE(t) \quad \text{---------- (4.3)}$$

The matrix G is a feedback gain matrix, the (arbitary) selection of which determines the characteristic of the observer if the system is observable, Kailath [86]. The poles of the observer process are given by the matrix (A - GC).

The above equation can be carried over to decentralised estimation if the system is partitioned into its **cyclic** and **acyclic** components. Because each cyclic (and acyclic) sub-system is dynamically decoupled from each other, as shown earlier, the assignment of poles to each sub-observer can be achieved independently. An acyclic sub-system is not considered in the thesis since it is a trivial sub-system, i.e. it consists of one element only.

The estimation process of a sub-plant can be found by using the following equations

$$\dot{X}_i(t) = AX_i(t) + B_iU_i(t) + \left(\sum_{j=1}^{s} A_{ij}X_j(t) \right)$$

$$\text{---------- (4.4)}$$

$$Y_i(t) = C_iX_i(t)$$

$$X_i(t) \in \Re^{ni}$$

$$Y_i(t) \in \Re^{pi}$$

$$U_i(t) \in \Re^{mi}$$

Where, $\sum_{i=1}^{s} n_i = N$, $\sum_{i=1}^{s} m_i = M$ and $\sum_{i=1}^{s} p_i = P$

s = number of partitions or sub-systems. And all pairs $[A_i, C_i]$ are potentially observable.

The error between the sub-plant and the estimation model output can be determined i.e.

$$E_i(t) = Y_i(t) - C_i\hat{X}_i(t) \quad \text{---------- (4.5)}$$

This error vector can be used again to improve the estimation process,

$$\dot{\hat{X}}_i(t) = A_i\hat{X}_i(t) + B_iU_i(t) + (\sum_{j=1}^{s} A_{ij}\hat{X}_j(t)) + G_iE_i(t) \quad \text{----} \quad (4.6)$$

$$\hat{X}_i(t) = (A_i - G_iC_i)\hat{X}_i(t) + B_iU_i(t) + (\sum_{j=1}^{s} A_{ij}\hat{X}_j(t)) + G_iY_i(t) \quad \text{--} \quad (4.7)$$

Again the characteristic equation can be determined for the process of decentralised observation by the equation below:-

$$|\lambda I_i - (A_i - G_iC_i)| = 0 \quad \text{----------} \quad (4.8)$$

where I is an identity matrix and λ is a complex variable.

Let us consider a few applications of the decentralised estimation technique, using the above equations to show how the decentralised observers can be synthesised without checking for the stability of the interconnected sub-system due to the way the system is partitioned. This applies to the structural partitioning and any further quantitative decoupling by sensitivity analysis.

The two examples demonstrate how a system can be observed in a decentralised fashion by using two sub-observer processes executing concurrently, asynchronously and communicating only necessary state vector information between them at the start of a new computation cycle. In the first example, the system is partitioned structurally, into its **cyclic** and **acyclic** components. The second example demonstrates that a **cyclic** component of 4 by 4 order, could be further partitioned using the quantitative analysis as mentioned earlier in the section 4.2.1.2. The second example is a sub-problem of a large order, given in [87], where the 12 by 12 order was structually partitioned into 4 **cyclic** sub-sytems [7], where two sub-systems were of the order 4 by 4 and the other two of the order 2 by 2.

EXAMPLE 1

This example is taken from Orr's thesis [88], where the state

space representation of the plant is shown below:-

$$A = \begin{bmatrix} -4 & 7 & -1 & 13 \\ 0 & 3 & 0 & 2 \\ 4 & 7 & -4 & 8 \\ 0 & -1 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \\ 2 \\ -2 \end{bmatrix} \quad C^t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

By using the technique described in section 2.2.3 to expose the **cyclic** and **acyclic** part of the system, the rearranged representation of the plant is shown below:-

Note: the matrix A, B and C above are rearranged by partitioning to ${}^*A = P^tAP$, ${}^*B = P^tB$ and ${}^*C^t = C^tP$, respectively. However to simplify the notation in this example, the primes on the appropriate matrices are omitted.

$$A = \left[\begin{array}{cc|cc} -4 & -1 & 7 & 13 \\ 4 & -4 & 7 & 8 \\ \hline 0 & 0 & 3 & 2 \\ 0 & 0 & -1 & 0 \end{array}\right] \quad B = \left[\begin{array}{c} 0 \\ 2 \\ \hline 1 \\ -2 \end{array}\right] \quad C^t = \left[\begin{array}{cc} 1 & 0 \\ 0 & 0 \\ \hline 0 & 1 \\ 0 & 0 \end{array}\right]$$

The above partitioning technique shows that there are two sub-systems or processes in the plant, dynamically decoupled from each other. These sub-systems are as follows

$$A_1 = \begin{bmatrix} -4 & -1 \\ 4 & -4 \end{bmatrix} \quad A_2 = \begin{bmatrix} 3 & 2 \\ -1 & 0 \end{bmatrix}$$

where the eigenvalues for the system or sub-systems are as follows:-

$$\lambda_{11} = -4 + j2$$
$$\lambda_{12} = -4 - j2$$
$$\lambda_{21} = 2$$
$$\lambda_{22} = 1$$

The interacting sub-system for the above plant is the one shown:-

$$A_{12} = \begin{bmatrix} 7 & 13 \\ 7 & 8 \end{bmatrix}$$

The observation vectors for the two sub-systems are as follows:-

$$C_1 = [1\ 0] \qquad C_2 = [1\ 0]$$

To achieve the asymptotic stability for the observation processes **equation** 4.8 must be used. The eigenvalues can be chosen arbitrarily as explained earlier in this section, therefore the eigenvalues for both estimation processes were chosen as -5 and -4.

Then the observer poles for sub-system 1, can be found from

$$[A_1] - ([G_1][C_1]) = \begin{bmatrix} -4 & -1 \\ 4 & -4 \end{bmatrix} - \begin{bmatrix} g_{11} \\ g_{12} \end{bmatrix} [1\ \ 0]$$

which has a characteristic equation, calculated from the determinant of $\lambda I_1 - (A_1 - G_1C_1)$.

$$\lambda^2 + (8 + g_{11})\lambda + 20 + 4g_{11} - g_{12} = 0$$

To achieve the above objective the elements of the feedback matrix G_1, g_{11} and g_{12} are chosen to be 1 and 4 respectively. Similarly, for the other sub-system to achieve the identical dynamics required, the elements of the feedback matrix G_2, g_{21} and g_{22} are chosen to be 12 and 9 respectively. The resulting behaviour of a system as described by the estimated states $\hat{X}(t)$ when subjected to a step input, compared to the actual states $X(t)$, are shown in **fig.** 4.1(a) - (d). The results were obtained by simulation using an in-house simulation package (Interactive Simulation Language) in which system variables are designated as outputs from blocks.

The initial condition of the actual states was set up as follows:-

Block 1 (state X_4) is 1
Block 2 (state X_3) is -1
Block 3 (state X_2) is 5
Block 4 (state X_1) is -5

where as the initial condition for estimated states $\hat{X}_4$ - $\hat{X}_1$ were set at 0, these were labelled as block 20,19,14,15 respectively.

EXAMPLE 2

Evans [7] used the Gust-Alleviation Problem from [87] to illustrate a decentralised control scheme. The extract below is a sub-system of the Longitudinal motion of an aircraft's dynamics achieved by the structural partitioning technique [7]. This sub-system can be considered as a **system** due to the decomposition technique used.

$$A = \begin{bmatrix} 0.00 & 1.000 & 0.00 & 0.000 \\ -408.86 & -2.679 & -10.71 & -0.518 \\ 0.00 & 0.000 & 0.00 & 1.000 \\ -1.24 & -0.176 & -390.10 & -0.474 \end{bmatrix} \quad B = C^t = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

By using the **partitioning technique** of section 2.2.3, the above system is shown to be **strongly coupled**, i.e. the system is **cyclic**. However, using the **sensitivity matrix (quantitative) analysis**, the system can be further decomposed into two sub-systems [7], one with states {1, 2} and the other with states {3, 4}. These sub-systems are shown to be dynamically decoupled and are as follows:-

$$A_1 = \begin{bmatrix} 0.00 & 1.000 \\ -408.86 & -2.679 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0.00 & 1.000 \\ -390.10 & -0.474 \end{bmatrix}$$

where the eigenvalues for the system are:-

$$\lambda_{11} = -1.36 + j20.19$$
$$\lambda_{12} = -1.36 - j20.19$$
$$\lambda_{21} = -0.22 + j19.73$$
$$\lambda_{22} = -0.22 - j19.73$$

The interacting sub-systems for the two sub-systems are:

$$A_{12} = \begin{bmatrix} 0.00 & 0.000 \\ -10.71 & -0.518 \end{bmatrix} \quad A_{21} = \begin{bmatrix} 0.00 & 0.000 \\ -1.24 & -0.176 \end{bmatrix}$$

The observation vectors for the two sub-systems are as follows:-

$$C_1 = C_2 = [1 \quad 0]$$

To achieve state estimation of the system, decentralised observers are employed on the partitioned sub-systems. To achieve the asymptotic stability for the observer processes, **equ.** 4.8 must be used; hence the eigenvalues can be chosen arbitrarily.

Two equations for the estimation processes are shown below for sub-system 1 and 2 respectively.

Estimation 1

The observer poles for sub-system 1 can be found from

$$[A_1] - ([G_1][C_1]) = \begin{bmatrix} 0.00 & 1.000 \\ -408.86 & -2.679 \end{bmatrix} - \begin{bmatrix} g_{11} \\ g_{12} \end{bmatrix} [1 \; 0]$$

the characteristic equation can be calculated from the determinant of $\lambda I_1 - (A_1 - G_1C_1)$.

The values of g_{11} and g_{12} can be chosen simply as 1 and 0 respectively. This gain factor will give the dynamics of the observer process as $-1.84 \pm j20.20$.

Estimation 2

The observer poles for sub-system 2 can be found from

$$[A_2] - ([G_2][C_2]) = \begin{bmatrix} 0.00 & 1.000 \\ -390.10 & -0.474 \end{bmatrix} - \begin{bmatrix} g_{21} \\ g_{22} \end{bmatrix} [1 \; 0]$$

By choosing the value of g_{21} and g_{22} as 3 and 0 respectively, the dynamics of this observer process will be $-1.74 \pm j19.71$. The resulting behaviour of the estimated states for a step input, compared to the actual states, is depicted in **fig.** 4.2 (a) - (d).

The initial conditions of the actual states were set up as follows:-

Block 3 (state x_1) is 2

Block 4 (state x_2) is -1

Block 5 (state x_3) is 0

Block 6 (state x_4) is -5

where as the initial conditions for the estimated states $\hat{x}_1$- $\hat{x}_4$ were set at 0, these were labelled as block 9 - 12 respectively.

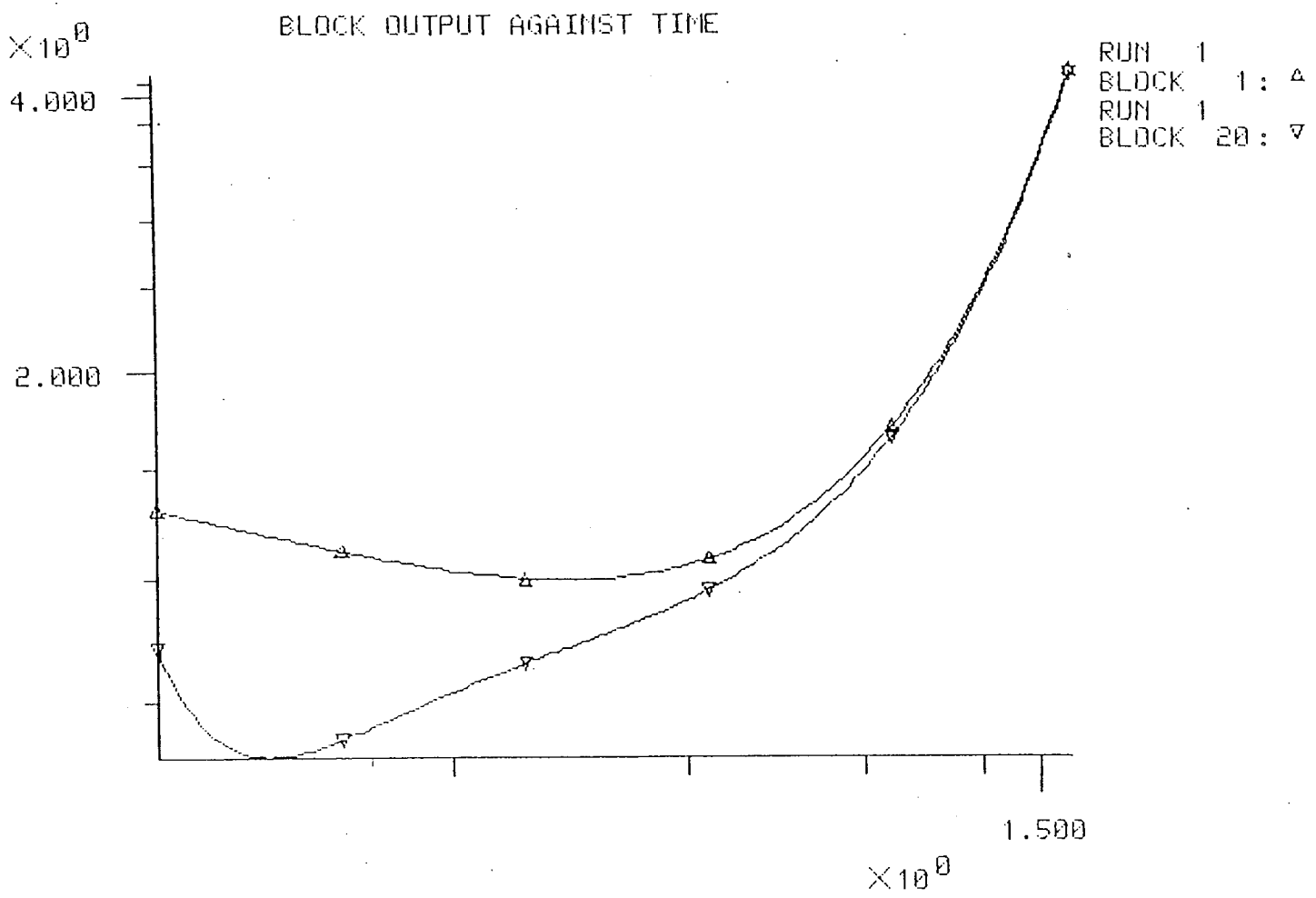


Figure 4.1 (a) State X_4 (block 1) and $\hat{X}_4$ (block 20) against time

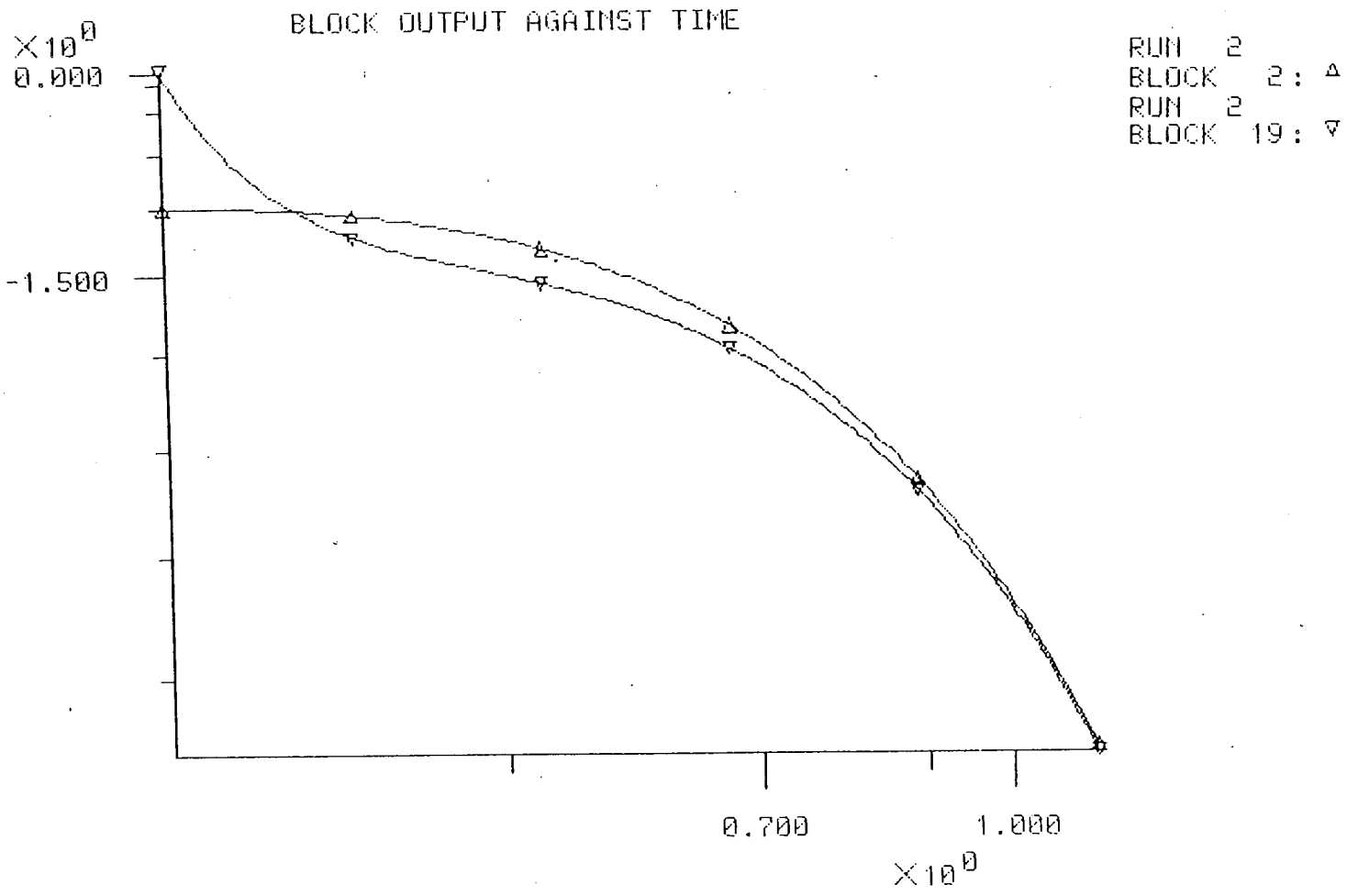


Figure 4.1 (b) State X_3 (block 2) and $\hat{X}_3$ (block 19) against time

LIBRARY AND INFORMATION SERVICES

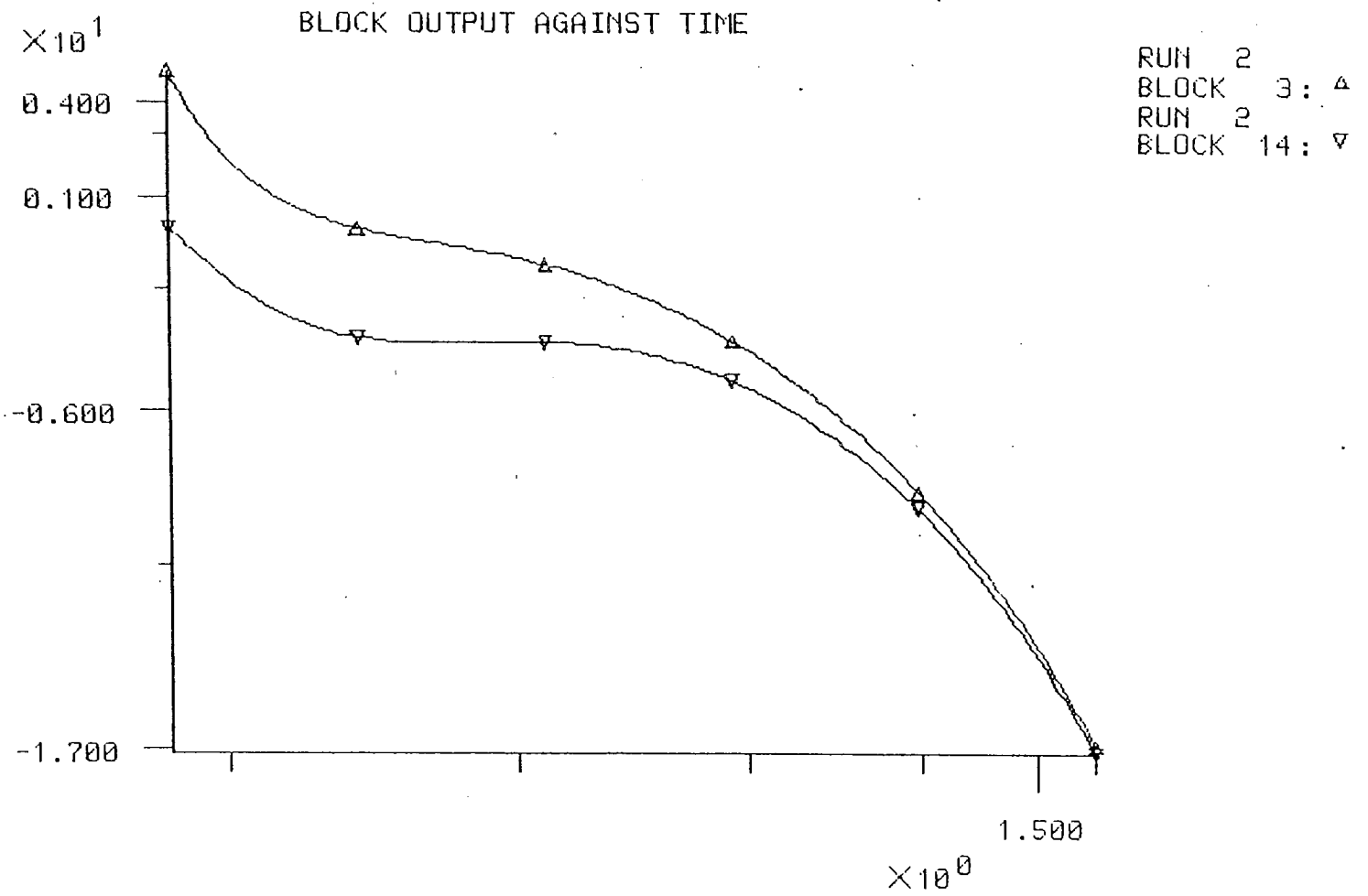


Figure 4.1 (c) State X_2 (block 3) and $\hat{X}_2$ (block 14) against time

LIBRARY
UNIVERSITY OF MANCHESTER INSTITUTE OF SCIENCE AND TECHNOLOGY

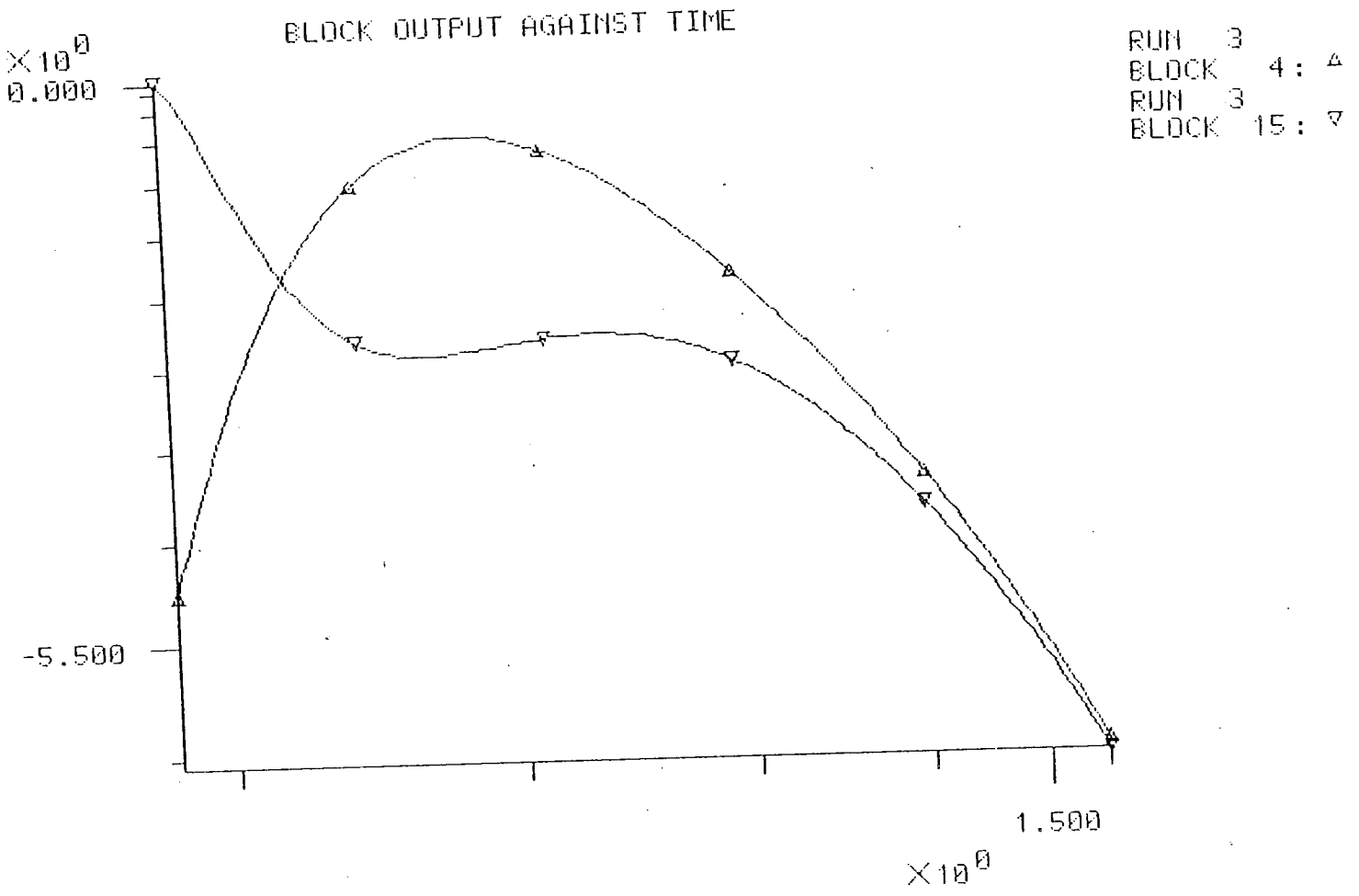


Figure 4.1 (d) State X_1 (block 4) and $\hat{X}_1$ (block 15) against time

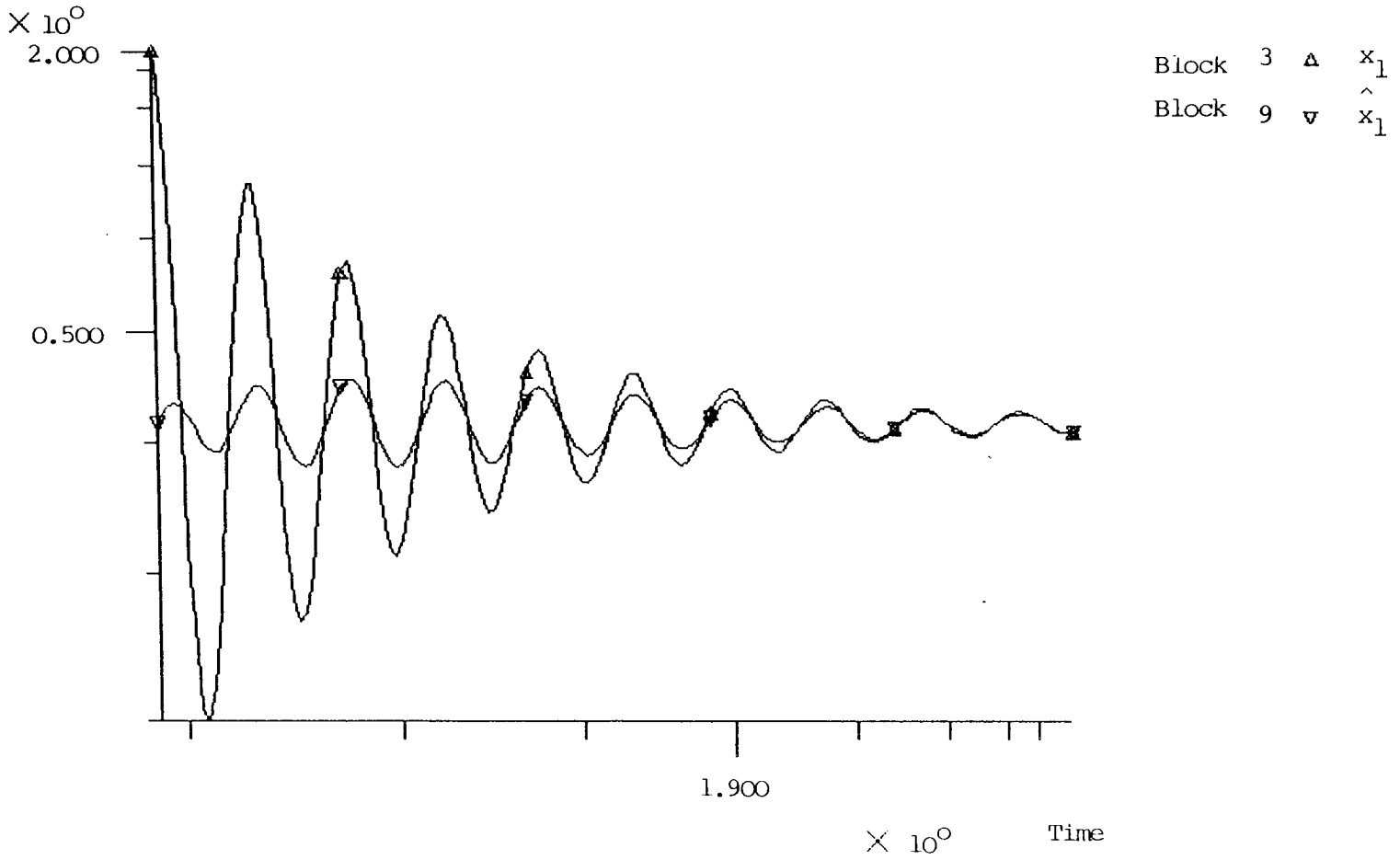


Figure 4.2 (a) State X_1 (block 3) and $\hat{X}_1$ (block 9) against time

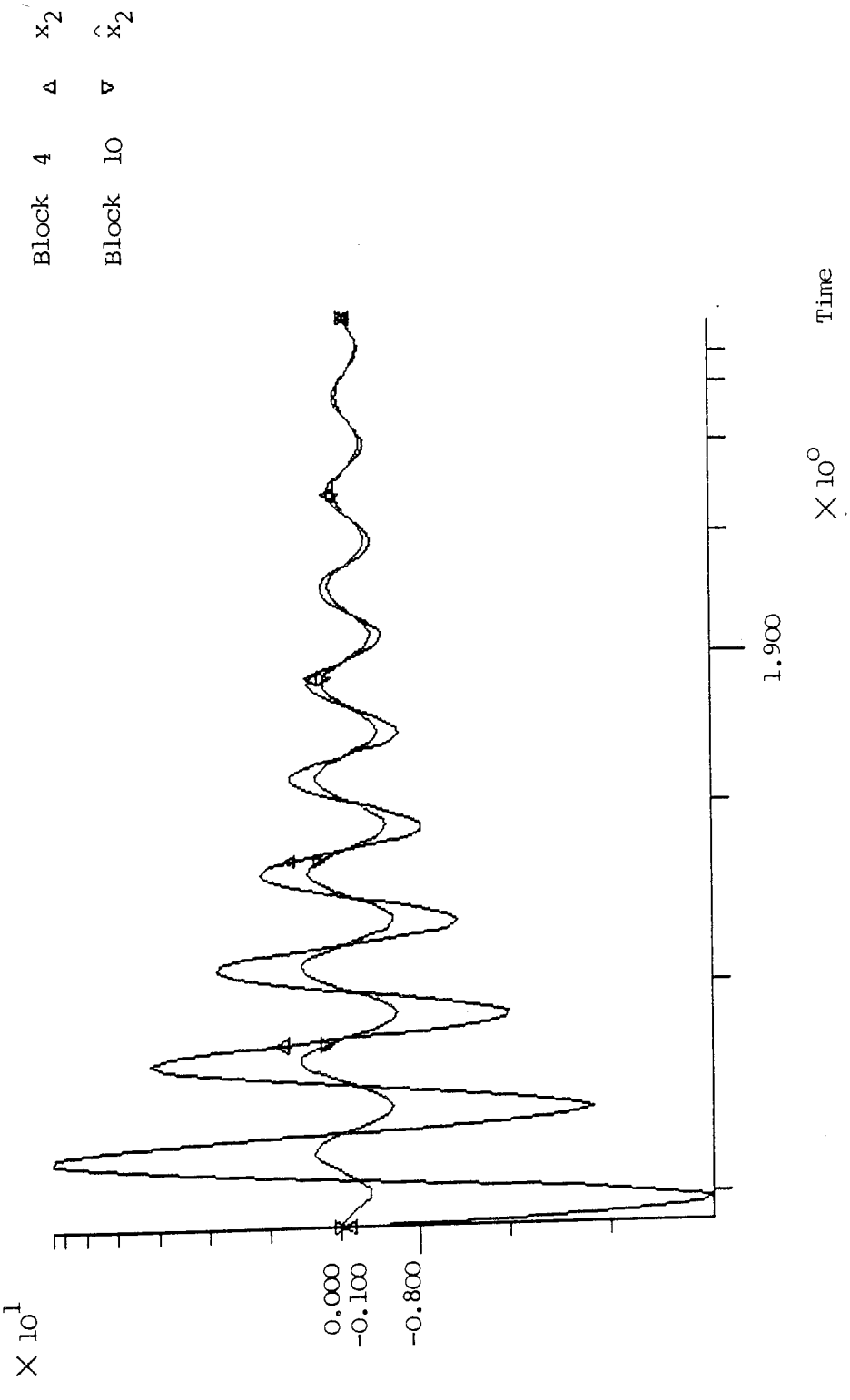


Figure 4.2 (b) State X_2 (block 4) and $\hat{X}_2$ (block 10) against time

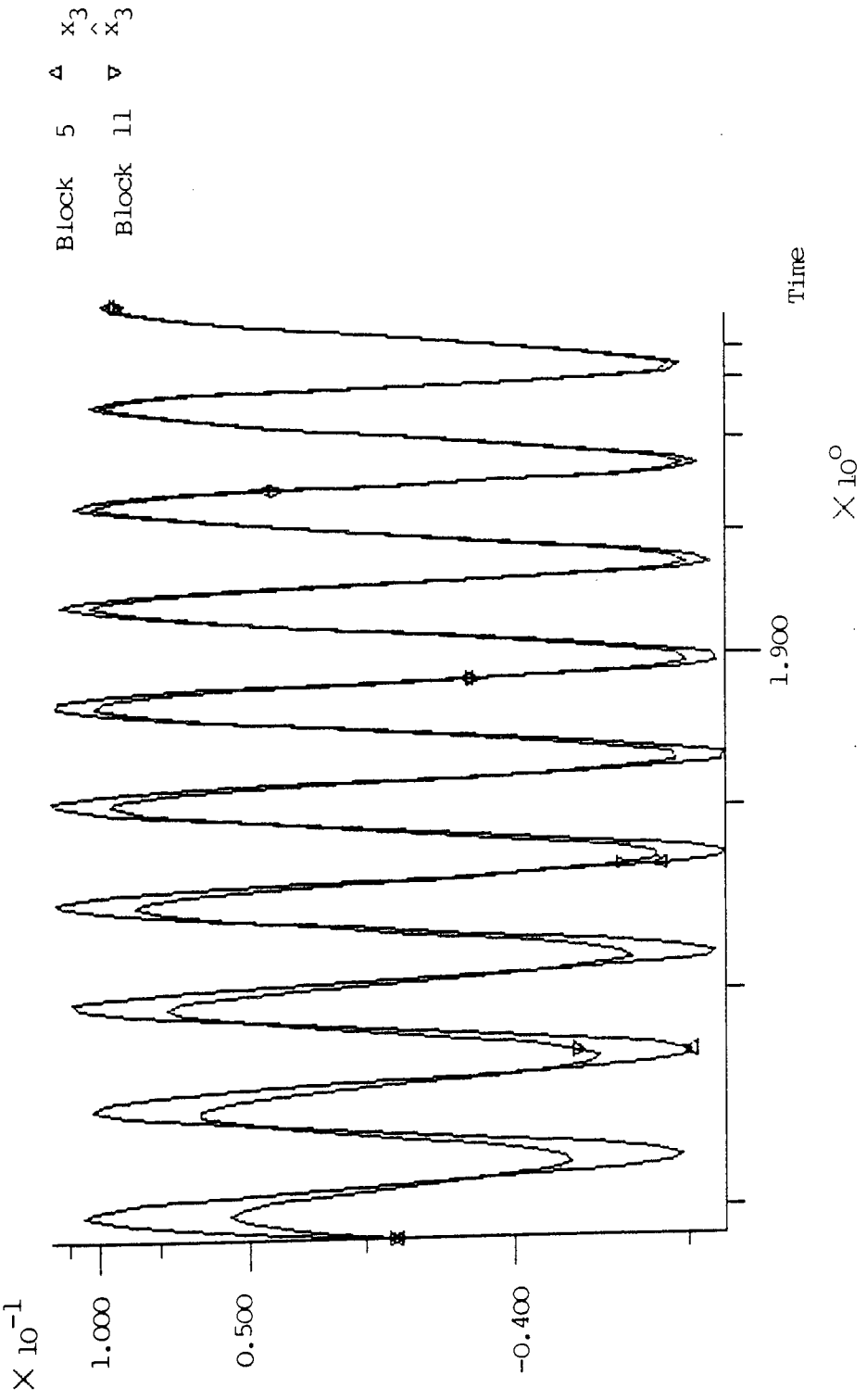


Figure 4.2 (c) State X_3 (block 5) and $\hat{X}_3$ (block 11) against time

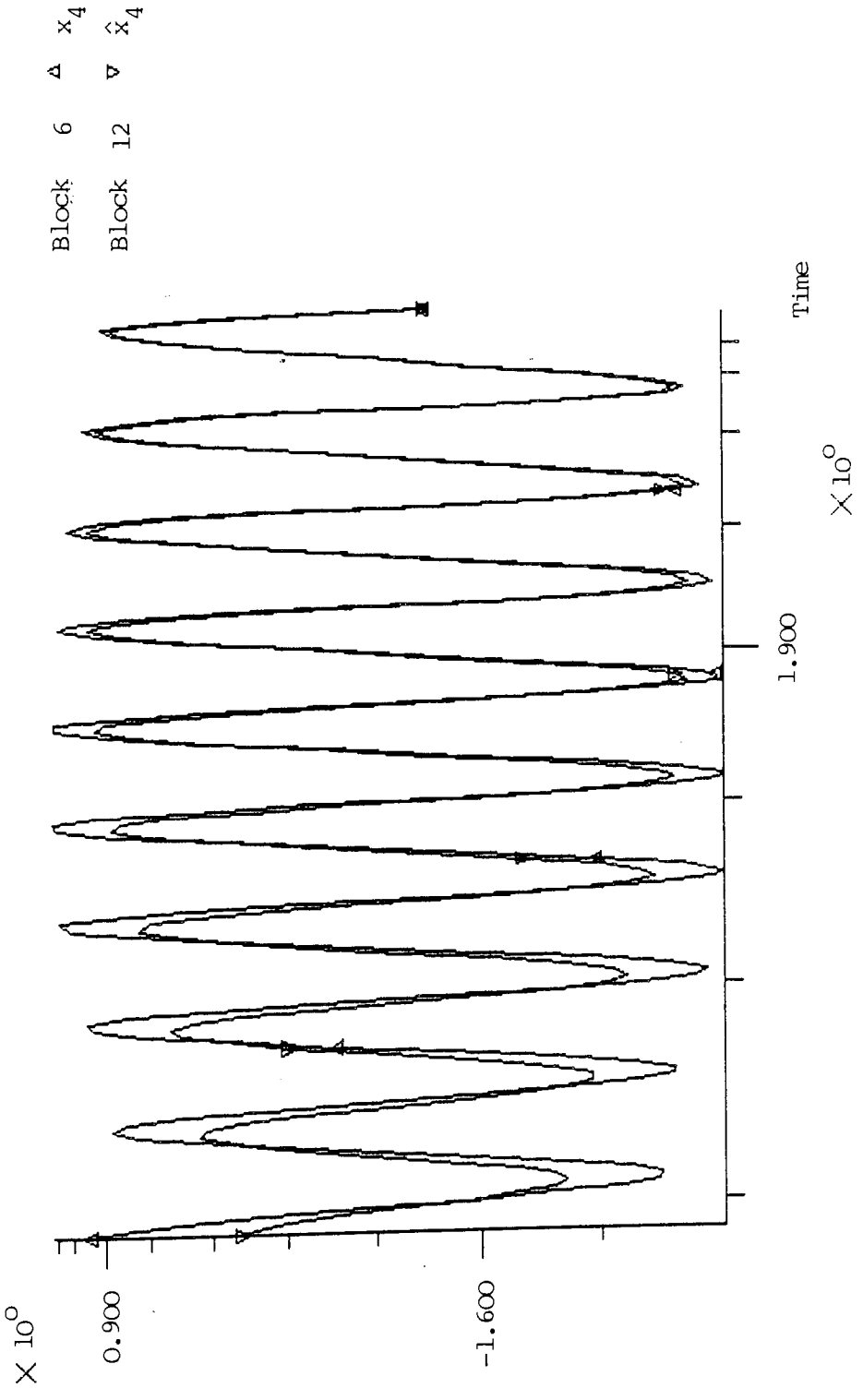


Figure 4.2 (d) State X_4 (block 6) and $\hat{X}_4$ (block 12) against time

4.2.1.4 Control System Containing Observers

Whatever the control design objective, emphasis must be placed on the stability of the system by the feedback of a known or measurable state vector. However, once an observer has been employed to obtain an estimate of the state vector and this estimated state vector has been used in place of its actual value, then the closed loop system is no longer dependent on the true state vector, but rather is characterised by its estimated version.

It is important and of interest therefore, to show that the **separation property** holds for the decentralised estimation proposed in this chapter; that is, to show that the eigenvalues of the observer and the closed-loop plant matrix can be **assigned separately**. The problem becomes whether, when a stable asymptotic observer is applied to an otherwise stable control system design, the overall closed loop system remains stable for any type of observer within this category?

In order to answer this question for the decentralised case, consider again the state **equation** 4.4 along with a newly defined control input signal:

$$U_i(t) = r_i(t) - F_i X_i(t) \quad \text{----------} \quad (4.9)$$

where F_i is a feedback matrix and $r_i(t)$ is a reference input for the sub-plant i of the appropriate dimension. Assuming the state vector to be known it follows on substitution of (4.9) into (4.4) that the closed loop eigenvalues are given by

$$\dot{X}_i(t) = (A_i - F_i)X_i(t) + \left(\sum_{j=1}^{s} A_{ij}X_j(t)\right) + B_i r_i(t) \text{----} (4.10)$$

However when an observer is used in order to obtain an estimate of the state vector, such that $X_i(t)$ becomes $\hat{X}_i(t)$ in (4.9),

the closed loop eigenvalues are found by means of the two equations

$$\dot{X}_i(t) = A_i X_i(t) - B_i F_i \hat{X}_i(t) + \left(\sum_{j=1}^{s} A_{ij} X_j(t)\right) + B_i r_i(t) \quad (4.11)$$

and

$$\dot{\hat{X}}_i(t) = G_i C_i X_i(t) + (A_i - B_i F_i - G_i C_i)\hat{X}_i(t) + \left(\sum_{j=1}^{s} A_{ij} \hat{X}_j(t)\right) + B_i r_i(t) \quad (4.12)$$

where the latter comes from the general observer **equation** (4.3).

Now if the composite closed loop system consists of interconnected sub-plant and observers, the two equations for all the sub-plants can be cast into a matrix form:

$$\begin{bmatrix} \dot{X}_1(t) \\ \dot{X}_2(t) \\ \vdots \\ \dot{X}_s(t) \\ \hline \dot{\hat{X}}_1(t) \\ \dot{\hat{X}}_2(t) \\ \vdots \\ \dot{\hat{X}}_s(t) \end{bmatrix} = \left[\begin{array}{ccc|ccc} A_1 & \cdots & A_{1s} & -B_1F_1 & \cdots & 0 \\ A_{21} & A_2 & \cdots A_{2s} & 0 & -B_2F_2 & \cdots 0 \\ \cdots & & & \cdots & & \\ A_{s1} & \cdots & A_s & 0 & \cdots & -B_sF_s \\ \hline G_1C_1 & 0 & \cdots 0 & A_1-G_1C_1-B_1F_1 & \cdots & A_{1s} \\ 0 & G_2C_2 & 0 \cdots 0 & \cdots A_2-G_2C_2-B_2F_2 & \cdots & A_{2s} \\ \cdots & & & \cdots & & \\ 0 & \cdots & G_sC_s & A_{s1} & \cdots & A_s-G_sC_s-B_sF_s \end{array}\right] \times \begin{bmatrix} X_1 \\ X_2 \\ \cdots \\ X_s \\ \hline \hat{X}_1 \\ \hat{X}_2 \\ \cdots \\ \hat{X}_s \end{bmatrix} + \begin{bmatrix} B_1r_1 \\ B_2r_2 \\ \cdots \\ B_sr_s \\ \hline B_1r_1 \\ B_2r_2 \\ \cdots \\ B_sr_s \end{bmatrix}$$

By adding column $s+i$ to column i and subsequently subtracting row i from row $s+i$, for all $i = 1$ to s, the above complicated matrix can be reconfigured as

$$\left[\begin{array}{ccc|ccc} A_1-B_1F_1 & \cdots & A_{1s} & -B_1F_1 & \cdots & 0 \\ \cdots & & & \cdots & & \\ A_{s1} & \cdots & A_s-B_sF_s & 0 & \cdots & -B_sF_s \\ \hline 0 & \cdots & 0 & A_1-G_1C_1 & A_{12} \cdots & A_{1s} \\ \cdots & & & \cdots & & \\ 0 & \cdots & 0 & A_{s1} & A_{s2} \cdots & A_s-G_sC_s \end{array}\right]$$

From the above matrix it can be seen that the composite system is made up of two **cyclic** sub-systems. Therefore feedback controllers and observers are dynamically decoupled. Hence the controller can be designed as though the actual state vector is available, and the

observer can be designed without reference to the type of feedback control action, if any. This feature is known as the Separation property.

4.2.2 Secondary communication

This section systematically derives the **necessary communications** required for controlling a plant in a desired fashion. This technique identifies those state variables of other stations which interact with a given station. This extends the work of Momen et al [6] who identified the necessary communication to control a plant in a decentralised fashion but which ignored the state variables, (due to other stations), that affect the station.

The identification of those state variables that affect the station is necessary for estimating state variables of a given station by the observation process, as described in the previous sub-section, and it is also required by a controller of the same station needing to apply some feedback strategy.

The following definitions are required for this identification. Most were introduced in chapter 3.

Ru_i = Reachable space by control station i.

Ry_i = Observable " " " " "

K_i = Potential controllable space of station i.

M_i = " observable " " " "

$L_i = K_i \cap M_i^t$

L_i^* = Space extended by communication with the other station.

$^{*}L_i$ = Designated controllable & obs. space (if an overlap)

EL_i = Extended $^{*}L_i$ space.

S = Number of stations.

NOTE: The requirement here is that the distributed control stations can observe and can control the complete system. This implies a necessary condition for the complete system to be controllable and observable [6].

If a system consists of a set of interacting sub-sytems is to be controlled in a desired fashion, then it is necessary to account for the effect of the interaction between the sub-spaces of the other sub-systems. The interaction of the other sub-systems may or may not be desirable; in either case, the influencing states of the other sub-systems must be determined so that the appropriate action can be taken by the controlling station or processes. The identification of the interactions of the stations can be found by the steps described below. The information on these state variables which need to be communicated to the station i will be placed in the set EI_i.

The state variables, $^{*}I_i$, in the stations i which are affected by the other (stations) state variables are found by.

$$^{*}I_i = {}^{*}L_i \cap \left(\bigcup_{j=1}^{s} RU_j\right)$$

where $i \neq j$

If the set of such states is empty, then the sub-system i is completely decoupled from the other sub-systems; therefore no communication is required by the station i. However if the set is not empty then the steps below are necessary.

The state variables in other sub-systems which interact with sub-system i can be identified as follows:-

For each state variable in the set ${}^{*}I_i$ find its predecessors in the digraph of the system and form a set, P_i. This is the set of all state variables which cause a disturbance to the state variables in station i, including predecessor state variables which belongs to station i itself.

$$P_i = \{\text{states which are predecessors of } s \mid \forall\, s \wedge s \in {}^{*}I_i\}$$

The predecessor state variables, belonging to the other sub-systems can be found by eliminating from the set P_i all the elements which may belong to the local set ${}^{*}L_i$. The new formed set, ${}^{*}I_i{}^{*}$, contains all the states in other sub-systems which affect the state variables in sub-system i.

$$ {}^{*}_{i}{}^{*} = P_i \cap \overline{{}^{*}L_i} $$

The above equation finds all the state variables which affect the controllable space of station i. However, this does not consider the direct effect which the input vectors of all the sub-systems have on the state variables of the particular station i. These need to be determined because, implicitly, the set of input (control) vectors defines the controllable spaces of each sub-system.

It is therefore necessary to consider the input vectors as an extension of the system, that is, as an extension of the state space representation of the system. Therefore the modification to the above procedure is as follows:-

STEP 1 :

$$P_i = \{\text{states which are predecessors of } s \mid \forall\, s \wedge s \in {}^{*}I_i\}$$

$$EP_i = \{\text{states which are predecessors of } es, \text{ in the extended representation only} \mid \forall\, es \wedge es \in {}^{*}L_i\}$$

STEP 2 :

$^{*}EP_i = P_i \,||\, EP_i$

|| = means concatenate space in the boolean domain

STEP 3 :

$EL_i = {}^{*}L_i \,||$ {control vector associated with the sub-system}

Therefore,

STEP 4 :

$$EI_i = {}^{*}EP_i \cap \overline{EL_i}$$

The set EI_i will contain all state variables that need to be communicated to station i. To demonstrate the above procedure a simple example is given below:-

EXAMPLE

This is an adaptation of the power station boiler controller problem considered by Orr [89], to show the secondary communication procedure.

Note: Here one is concerned with the boolean domain only, and, using the concept of set theory discussed above, '1' means state is present, and '0' means state is not present in the set. Also an understanding of the potential controllability and observability concept is required, described in chapter 3.

$$A_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad B_b1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad B_b2 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad B_b3 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$C_b^t\,1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad C_b^t\,2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad C_b^t\,3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

From the above state space representation, the following sets

are found. Using the methods described in chapter 3 for constructing these sets.

$$R_{U1} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad R_{U2} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \qquad R_{U3} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$R^{t}_{Y1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad R^{t}_{Y2} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \qquad R^{t}_{Y3} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$K_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad K_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \qquad K_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$M^{t}_{1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad M^{t}_{2} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \qquad M^{t}_{3} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$L_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad L_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad L_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Now under a decentralised structure the union of all the common sub-space sets must be a unit vector, (see chapter 3) i.e. union of L_1, L_2, L_3 must be a unit vector. Therefore no communication is required to make the given **system** controllable.

Therefore the sub-space (L_1, L_2, L_3) normally extended by communication remains the same:-

$$L^{*}_{1} = L_1 \qquad L^{*}_{2} = L_2 \qquad L^{*}_{3} = L_3$$

The above procedure so far is based upon the work of Momen et al [6]; in this thesis it is termed **primary communication**. However, further communication is required for estimating the values of the state variables by the observation process; this is introduced in this thesis and termed **Secondary communication**.

From the above sets it can be seen that state variable 2 is controlled by L_2 and L_3. Hence, the load can be shared between the two stations. There is only one controllable space that can be designated to

control a particular sub-space and this is given by the set L^*_3. Therefore, the designated spaces are as follows:-

$$^*L_1 = L^*_1 \qquad ^*L_2 = L^*_2 \qquad ^*L_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The above sets show that there was no communication required by the stations. However, it is necessary to determine which are the interacting states between the designated controller spaces. These are determined for each controller using the above steps as follows and are labeled as **Process 1, 2** and **3** :

Process 1 (Stn. 1):

First find all the state variables that interact with this control station

$$^*I_1 = L_1 \cap (RU_2 \cup RU_3)$$

$$= [1\ 0\ 0]^t \cap ([1\ 1\ 0]^t \cup [1\ 1\ 1]^t)$$

$$= [1\ 0\ 0]^t \text{ (*i.e. state variable 1 is affected*)}$$

Note: The above set, $^*I_1$ shows that the state variable 1 is affected. However, It is interesting to see that there is no predecessor of state variable 1, i.e. no other state variable is agitating it. The agitation is due to the influence of the inputs from the other stations. This is why in the development of **secondary communication** identification theory, the input vectors were considered as an extension of the state space representation of the system.

Now, find all the predecessors for each of the state variables of the above set $^*I_1$, as described in the extended theory of secondary communication, i.e. **step 1** above.

The two sets which represent the interaction with the station 1 state variables due to state variables and control vectors of the system are shown in set P_1 and EP_1 respectively,

$$P_1 = [0\ 0\ 0]^t$$

$$EP_1 = [1\ 1\ 1]^t \text{ (*i.e. inputs } U_1, U_2 \text{ and } U_3*)$$

Note: that P_1 is an empty set and therefore the non-extended version would have indicated no interaction with this station. This is obviously wrong. These two spaces are concatenated for use later as indicated in **step 2**, above.

$$^*EP_1 = P_1 \,||\, EP_1 = [0\ 0\ 0\ 1\ 1\ 1]^t$$

The next step **(step 3)** is to find the actual or designated space, EL_1, which this station is to control. This is shown below,

$EL_1 = {}^*L_1 \,||$ {all the control vectors associated with the station or designated to it }

$$= [1\ 0\ 0]^t \,||\, [1\ 0\ 0]^t$$

$$= [1\ 0\ 0\ 1\ 0\ 0]^t$$

The final step is to eliminate from the set $^*EP_1$ all the state variables associated with station i, (as shown in **step 4**) which will leave in the set EI_1 only those state variables which do not belong to this station. Therefore, these state variables must be communicated to it, i.e.,

$$EI_1 = {}^*EP_1 \cap \overline{EL_1}$$

$$= [0\ 0\ 0\ 1\ 1\ 1]^t \cap [0\ 1\ 1\ 0\ 1\ 1]^t$$

$$= [0\ 0\ 0\ 0\ 1\ 1]^t$$

For station 1, communication was required from the other stations concerning the new computed control vectors U_2 and U_3. However, no interacting information was required concerning the state variables of the system.

Process 2 (stn. 2):

$$^{*}I_2 = L_2 \cap (RU_1 \cup RU_3)$$

$$= [0\ 1\ 0]^t \cap ([1\ 1\ 1]^t \cup [1\ 1\ 1]^t)$$

$$= [0\ 1\ 0]^t \text{ (*i.e. state 2 is agitated*)}$$

$$P_2 = [0\ 0\ 1]^t \text{ (*" " 3 affects state 2*)}$$

$$^{*}EP_2 = [0\ 0\ 1\ 1\ 1\ 0]^t$$

$$EI_2 = [0\ 0\ 1\ 1\ 0\ 0]^t$$

This shows it is necessary to communicate (the control vector U_1 and the state 3) to the **process 2**.

process 3 (Stn. 3):

$$^{*}I_3 = [0\ 0\ 1]^t$$

$$P_3 = [0\ 0\ 1]^t$$

$$^{*}EP_3 = [0\ 0\ 1\ 1\ 0\ 1]^t$$

$$EL_3 = [0\ 0\ 1\ 0\ 0\ 1]^t$$

$$EI_3 = [0\ 0\ 0\ 1\ 0\ 0]^t$$

Station 3 requires communication about the control vector U_1.

Fig. 4.3 shows a diagrammatic representation of the processes.

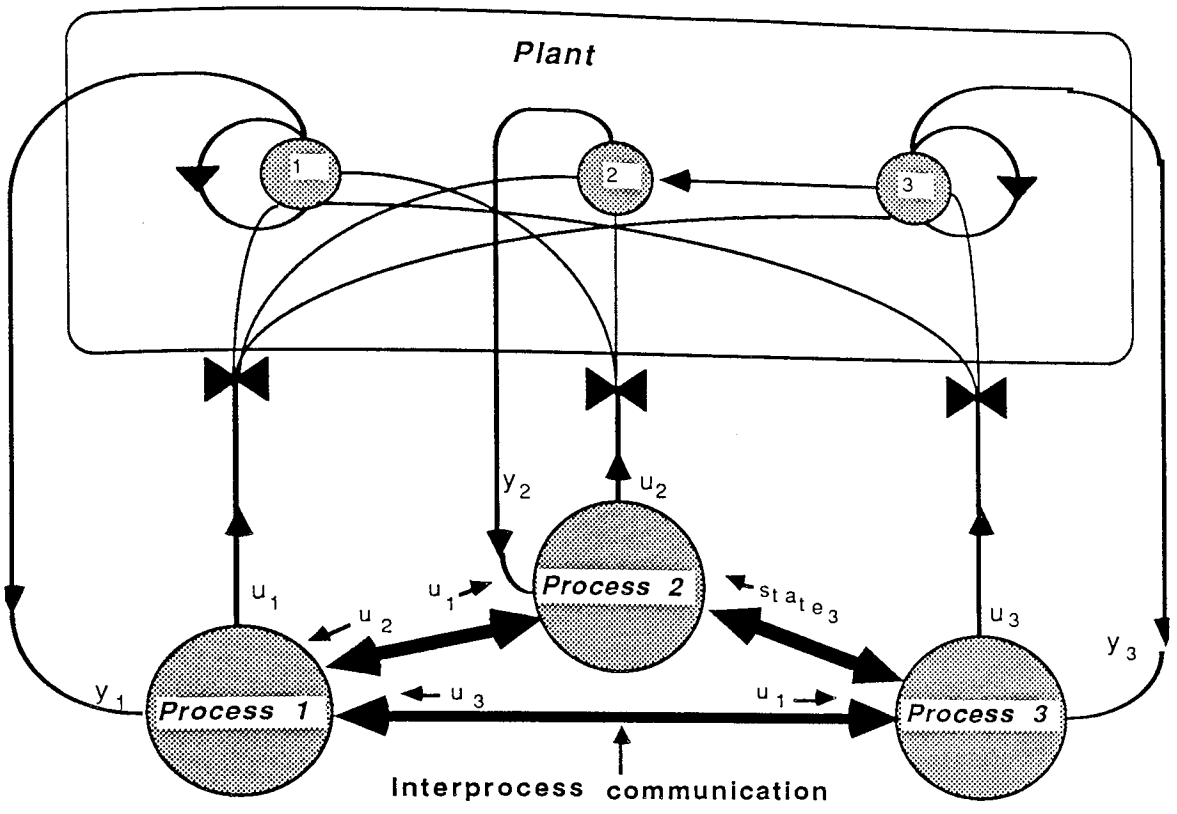


Figure 4.3 Secondary communications

4.2.3 Robust Decentralised Observer

The above new observation technique overcomes one aspect of the reliability problem, namely, the problem of stability, by requiring that each sub-plant observed is dynamically decoupled from every other. The technique still suffers from one problem. If there is a failure in one of the processes or in the communication network then no information transfer is possible and consequently the observer process will malfunction.

This problem is known classically as the **unknown input observer** problem. Here we present simple conditions under which **full** state estimation is possible without communications. The decomposition technique mentioned above allows one to consider each sub-system, as a **system** in its own right. The well-known procedure of the **Luenberger observer** is used to show the necessary design

constraints and the mechanism within this simulation process is available to the designer for minimizing the approximation discrepancy.

Consider the dynamical **system** described by

$$\dot{X}(t) = AX(t) + BU(t) + \ddot{E}D(t)$$

$$Y(t) = CX(t) \qquad \text{----------(4.13)}$$

Where the matrix $\ddot{E}$ is the interaction of the unknown input vector $D(t)$ on the system. The observation process for the above can be represented as

$$\dot{\hat{X}}(t) = {}^{*}F\hat{X}(t) + {}^{*}PU(t) + NY(t) \qquad \text{----------(4.14)}$$

where

$$\hat{X}(t) = TX(t) \qquad \text{----------(4.15)}$$

and ${}^{*}F$, ${}^{*}P$, N and T are the corresponding coefficient matrices of appropriate sizes. For proper approximation, the error in the estimation process must tend to zero as t approaches infinity i.e.

$$\dot{e}(t) = \dot{\hat{X}}(t) - T\dot{X}(t) = 0 \text{ as } t \longrightarrow \infty \qquad \text{----------(4.16)}$$

$$= ({}^{*}FT - TA + NC)X(t) + ({}^{*}P - TB)U(t) - (T\ddot{E})D(t)$$

Therefore, to achieve the full reconstruction of the state estimation, the following conditions must be satisfied

$${}^{*}FT = TA - NC \qquad \text{----------(4.17)}$$

$${}^{*}P = TB \qquad \text{----------(4.18)}$$

$$T\ddot{E} = 0 \qquad \text{----------(4.19)}$$

The solution of these coefficient and the transformation matrices ${}^{*}F$, ${}^{*}P$, N and T must be obtained so that a direct reconstruction of the state vector can be achieved. This is possible, provided that ${}^{*}F$ is a stable matrix and the transformation matrix is of full rank, and also that the original process is observable. This observability condition is required so that the system states can be accessed through the process outputs for reconstruction.

4.3 DISCRETE-TIME SYSTEM

4.3.1 Decentralised (Estimator) Kalman Filter

Many of the ideas developed for the Observer case in the continuous-time will be carried over into this section. However, there will be two important differences: the noise on the system will be taken into consideration and the system will be discrete-time rather than continuous-time. The main object behind filtering is to reduce the effect the noise or unwanted signal has on measurements obtained and on values calculated from those measurements.

The **Kalman filter** was developed by Kalman and Bucy [11] in the early **1960's**. An historical account of filtering in general is given in **Optimal Filtering** by Anderson and Moore [81]. The **Kalman filter** process is computed easily by digital computers because we are dealing with discrete-time systems.

In this section, a **one-step prediction** of the state estimator will be used to show that decentralised estimation can be computed if the system is **partitioned** using the technique shown earlier in this thesis. This will be demonstrated in the same way as the **Luenburger Observer** technique was developed for the decentralised case in continuous-time.

Consider a discrete version of (4.1), such that noise is allowed for, with the time index $k \geq 0$;

$$X(k+1) = AX(k) + BU(k) + DW(k)$$

and ---------- (4.20)

$$Y(k) = CX(k) + V(k)$$

$X(k)$ is the state vector at time instant k, $Y(k)$ is the output signal at the same time instant and is corrupted by measurement noise $V(k)$. The process $W(k)$ is also considered in the form of a noise

sequence although it is possible to regard it as a system input. The assumption made here is that $V(k)$ and $W(k)$ are independent, Gaussian, white processes with zero mean and finite covariance equal to R and Q respectively.

The filtering problem can then be considered as one in which a prediction (estimation) of the state vector must be obtained for time instant k at time instant $k-1$ using measured values taken up to and including that same time instant. Note, that this **one-step filtering** technique can be expanded for the more general **N-step** ahead predictor and various other assumptions can be made for noise (see references [80,81]).

The state estimation update vector can then be written as:

$$\hat{X}(k+1|k) = A\hat{X}(k|k-1) + BU(k) + K(Y(k)-C\hat{X}(k|k-1)) \quad --(4.21)$$

which is optimal in the sense of minimum variance estimation, and the Kalman gain is found from,

$$K = AP(k|k-1)C^t(CP(k|k-1)C^t + R)^{-1} \quad \text{----------} \quad (4.22)$$

with the covariance matrix:

$$P(k+1|k) = AP(k|k-1)A^t - KCP(k|k-1)A^t + DQD^t \quad \text{------(4.23)}$$

A proof of the set of equations (4.21) - (4.23) is given in **reference** [81], where it is shown that in **equ.** (4.23) the variance of the error between actual and estimated state vector, is known as the reconstruction error. The selection of K in (4.22) will minimize the mean square reconstruction error, hence leading to an optimal filter for **equ.** (4.21).

It should be noted that for the discrete-time model, **equ.** (4.20), the index $k \geq 0$. The initial conditions for the state vector and error covariance matrix must therefore be defined such that

$$X(0|-1) = \mathcal{E}\{X0\} = M_0$$

and $P(0|-1) = P_0$

where $\mathcal{E}\{X0\}$ is the expected value, before any measurements have been taken, with mean M_0, covariance P_0. The distribution of $\mathcal{E}\{X0\}$ is Gaussian and is independent of both the $V(k)$ and $W(k)$ sequences.

The control input $U(k)$ has no effect on either the **Kalman gain** K or on the update of $P(k|k-1)$ obtained from equation (4.22) and (4.23) respectively. This important fact must be remembered when the equation is extended to cater for the decentralised case.

Note: In [80,81] the interesting connection between estimated and predicted signal vectors was shown as

$$\hat{X}(k+1|k) = A\hat{X}(k)$$

This means that the given filtered signal $\hat{X}(k)$, the best estimate of the signal one step in the future ignores noise and assumes that the signal dynamics matrix A operates only on the estimate. Thus simultaneous filtering and prediction of vector signals can be obtained from the same filter if the structure of **fig.** 4.4 is used [80,81]. This structure is used in the simulation of two examples given below, with the result taken from the filtering tap.

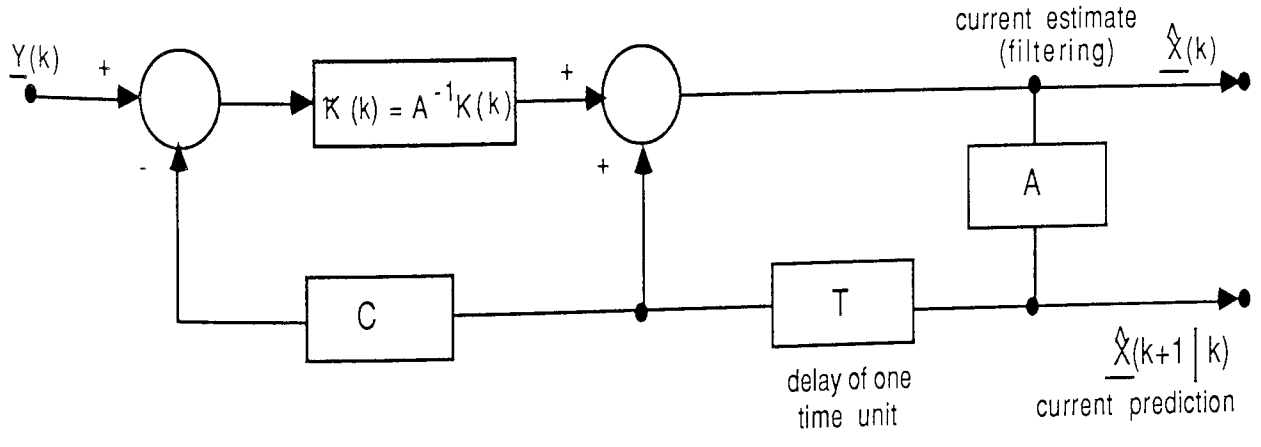


Figure 4.4 Simultaneous filtering and prediction of vector signals

If the system is partitioned into its **cyclic** and **acyclic** component sub-systems then the state of each sub-system can be estimated by communicating asynchronous Kalman filter processes, where the interactions from the other sub-systems can be treated as further control input vectors.

The state estimation update vector of the sub-system can then be written as

$$\hat{X}_i(k+1|k) = A_i\hat{X}_i(k|k-1) + K_i(Yi(k) - C_i\hat{X}_i(k|k-1)) + B_iU_i(k) + \left(\sum_{j=1}^{s} A_{ij}\hat{X}_j(k|k-1)\right) \quad (4.24)$$

where the Kalman gain is found from,

$$K_i = A_iP_i(k|k-1)C_i^t(C_iP_i(k|k-1)C_i^t + R_i)^{-1} \quad (4.25)$$

with the covariance matrix:

$$P_i(k+1|k) = A_iP_i(k|k-1)A_i^t - K_iC_iP_i(k|k-1)A_i^t + D_iQ_iD_i^t \quad (4.26)$$

To demonstrate that we can achieve state estimation via this parallel approach firstly, the previous example 2 of continuous-time section will be used with output noise considered only. Secondly, both input and output noise are considered with a very simple system. It can be seen from **equations** (4.24) - (4.26) that a considerable speed improvement will be achieved with this approach; for example matrix multiplication requires at least N^3 calculations (where N is the order of the matrix). The order of matrices in the examples used below is halved, therefore the speed off multiplication process is increased by 8.

EXAMPLE 1

The continuous-time representation of the system matrix, A_{cont} and the input matrix, B_{cont} of example 2 used in section 4.2.1.3 is mapped into the discrete-time representation by the transformation $A_{dis} = e^{A_{cont} \times T}$ for the discrete-time system matrix, and B_{dis}=

$A_{cont}^{-1}(A_{dis}-I)B_{cont}$ for the input matrix. Where I is the identity matrix and the output matrix, C is the same for both representations. Where T is chosen to be $^{1}/_{10}{}^{th}$ of the fastest mode.

The discrete-time representation is as shown below, for convenience the qualifying subscript $_{dis}$ is dropped from the matrices.

A_1 A_{12}

$$A = \left[\begin{array}{cc|cc} 0.138 & 0.4475e\text{-}01 & -0.8242e\text{-}02 & -0.1412e\text{-}02 \\ -18.30 & 0.1921e\text{-}01 & 0.7130e\text{-}01 & -0.3075e\text{-}01 \\ \hline 0.1341e\text{-}02 & -0.3595e\text{-}03 & 0.1282 & 0.4942e\text{-}01 \\ 0.8574e\text{-}01 & -0.6394e\text{-}02 & -19.27 & 0.1050 \end{array}\right]$$

A_{21} A_2

By decomposition of the above matrix A it can be shown that there are two sub-systems decoupled from each other represented by matrices A_1 and A_2 with their interacting matrices represented by A_{12} and A_{21} respectively. The input and output matrices for both sub-systems are:

$$B_1 = \begin{bmatrix} 0.5038e\text{-}01 & -0.3381e\text{-}03 \\ -0.8612 & -0.8242e\text{-}02 \end{bmatrix}$$

$$C_1 = C_2 = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} 0.6965e\text{-}05 & -0.5048e\text{-}03 \\ 0.1341e\text{-}02 & -0.8718 \end{bmatrix}$$

The resulting behaviour of the estimation in parallel approach is shown in **fig.** 4.5(a) while the sequential approach is shown in **fig.** 4.5(b). Only state 1 is shown to demonstrate that within a short time the results are similar. The implication is that the parallel approach is

approximately two hundred times faster in its estimation of the state vector without loss of accuracy in its computation. The standard deviation of the noise power on the output was 0.05.

EXAMPLE 2

The discrete-time representation for a simple system is shown below, for convenience the qualifying subscript $_{dis}$ is dropped from the matrices.

$$A = \begin{array}{c} \begin{matrix} A_1 & A_{12} \end{matrix} \\ \left[\begin{array}{c|c} 0.9512 & 0.1392 \\ \hline 0.0 & 0.9048 \end{array}\right] \\ \begin{matrix} A_{21} & A_2 \end{matrix} \end{array} \qquad B = \begin{array}{c} \begin{matrix} B_1 \end{matrix} \\ \left[\begin{array}{c} 0.4877e\text{-}01 \quad 0.3568e\text{-}02 \\ \hline 0.0 \qquad\qquad 0.4758e\text{-}01 \end{array}\right] \\ \begin{matrix} B_2 \end{matrix} \end{array}$$

$$C_1 = C_2 = [1]$$

By decomposition of the above matrix A it may be shown that there are two sub-systems decoupled from each other represented by matrices A_1 and A_2 and their interacting matrices represented by A_{12} and A_{21} respectively. The input and output matrices for both sub-systems are also shown.

The resulting behaviour of the estimation in the parallel approach is shown in **fig.** 4.6(a) while the sequential approach is shown in **fig.** 4.6(b). The result indicates that estimation in the parallel and sequential approaches is similar. Initially, in the parallel case, there is a glitch which is very quickly overcome by communications between the processes. However, the parallel approach is approximately eight times faster in its estimation of the state vector without loss of accuracy in its computation. The standard deviations of the noise power on input and output are 0.1 and 0.5 respectively.

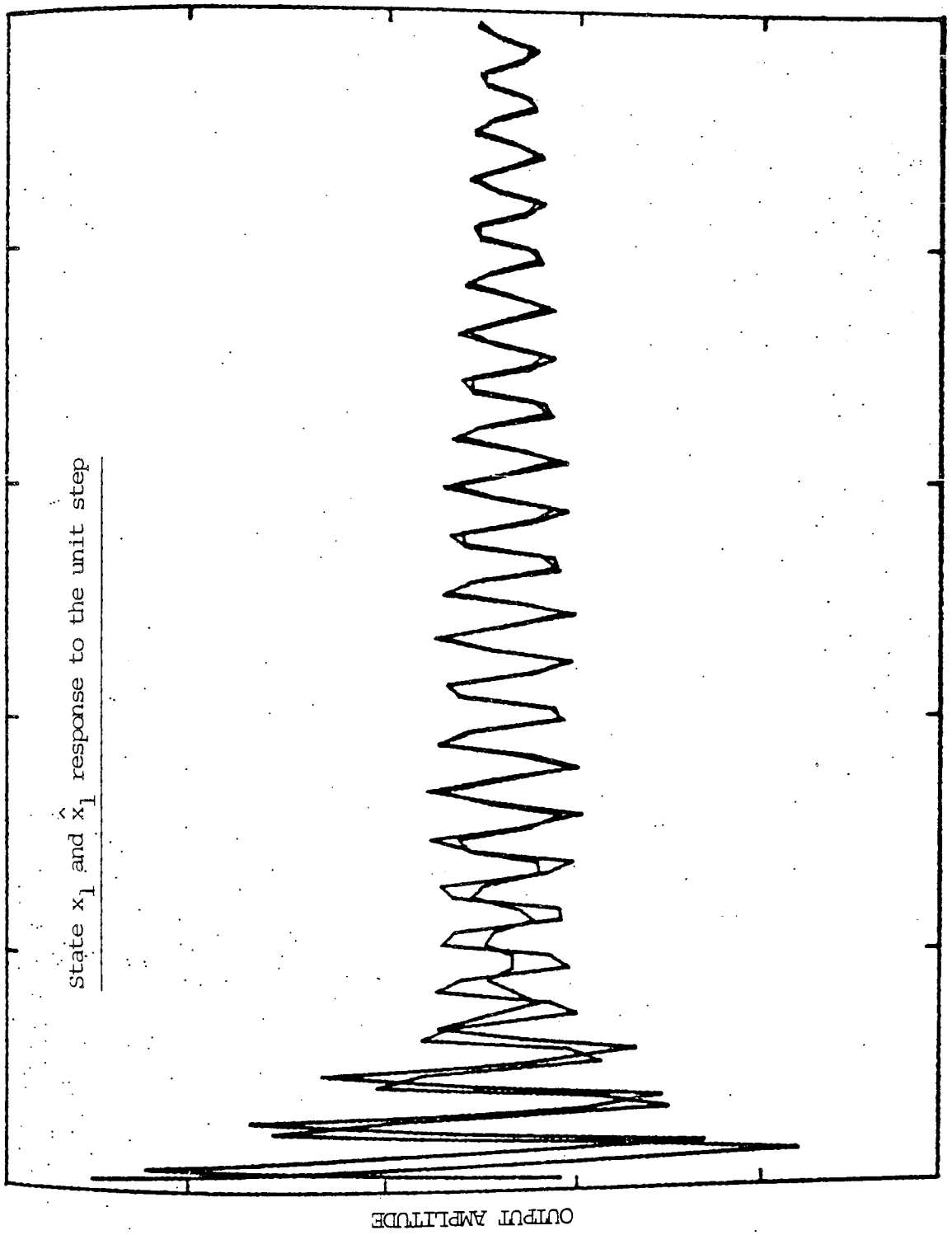


Figure 4.5 (a) State X_1 and $\hat{X}_1$ against time using parallel approach

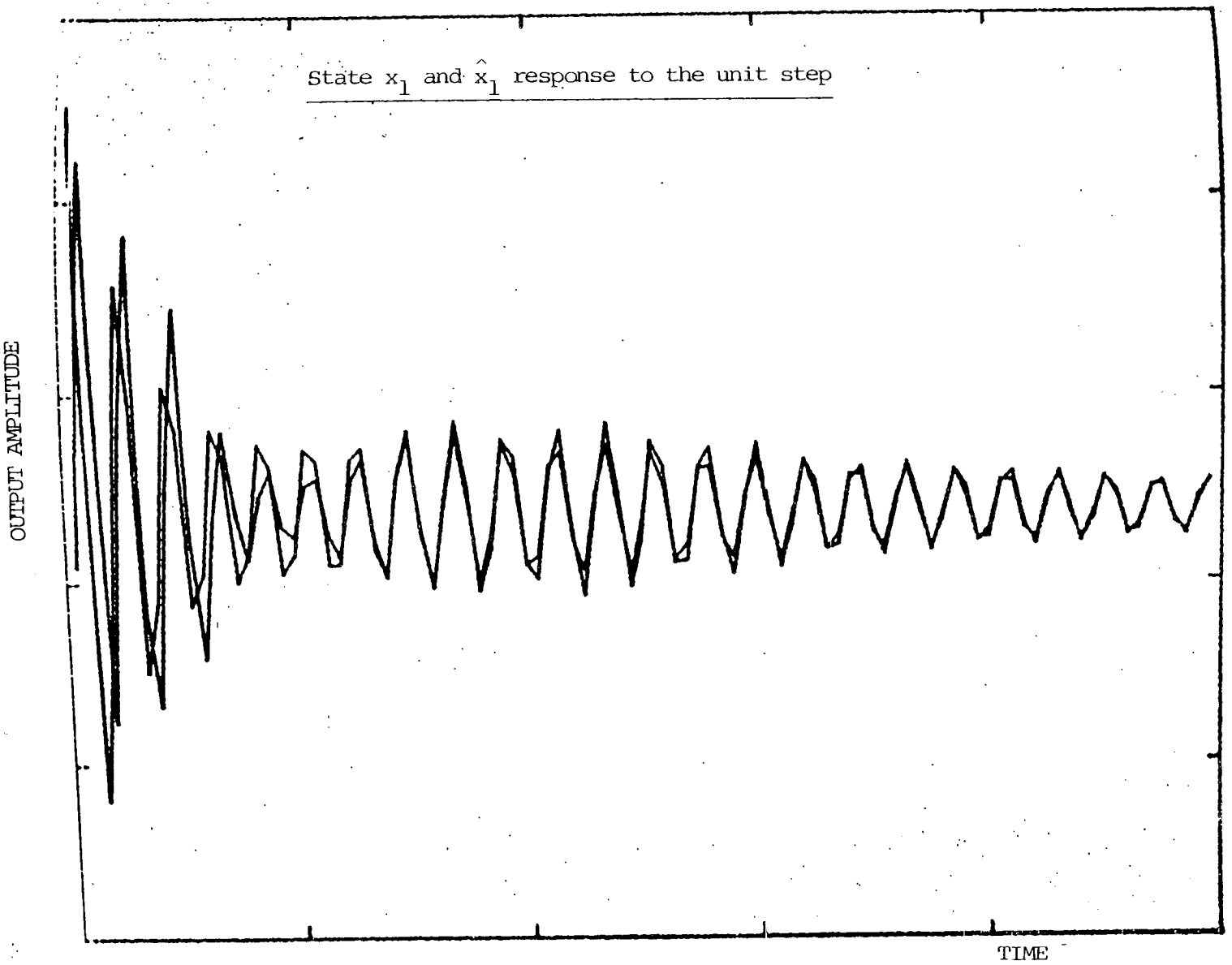


Figure 4.5 (b) State X_1 and $\hat{X}_1$ against time using standard approach

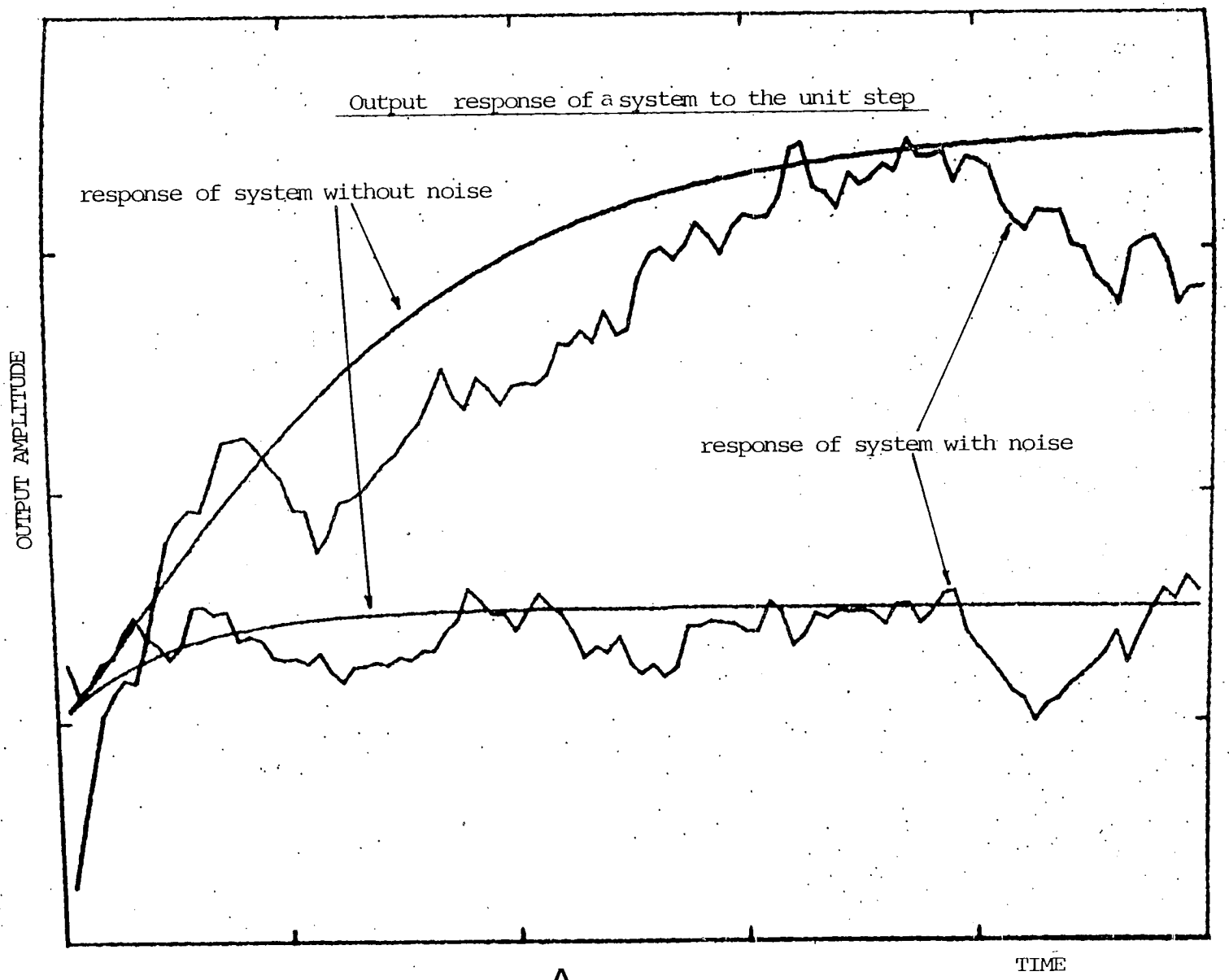


Figure 4.6 (a) State X_{1-2} and $\hat{X}_{1-2}$ against time using Parallel approach

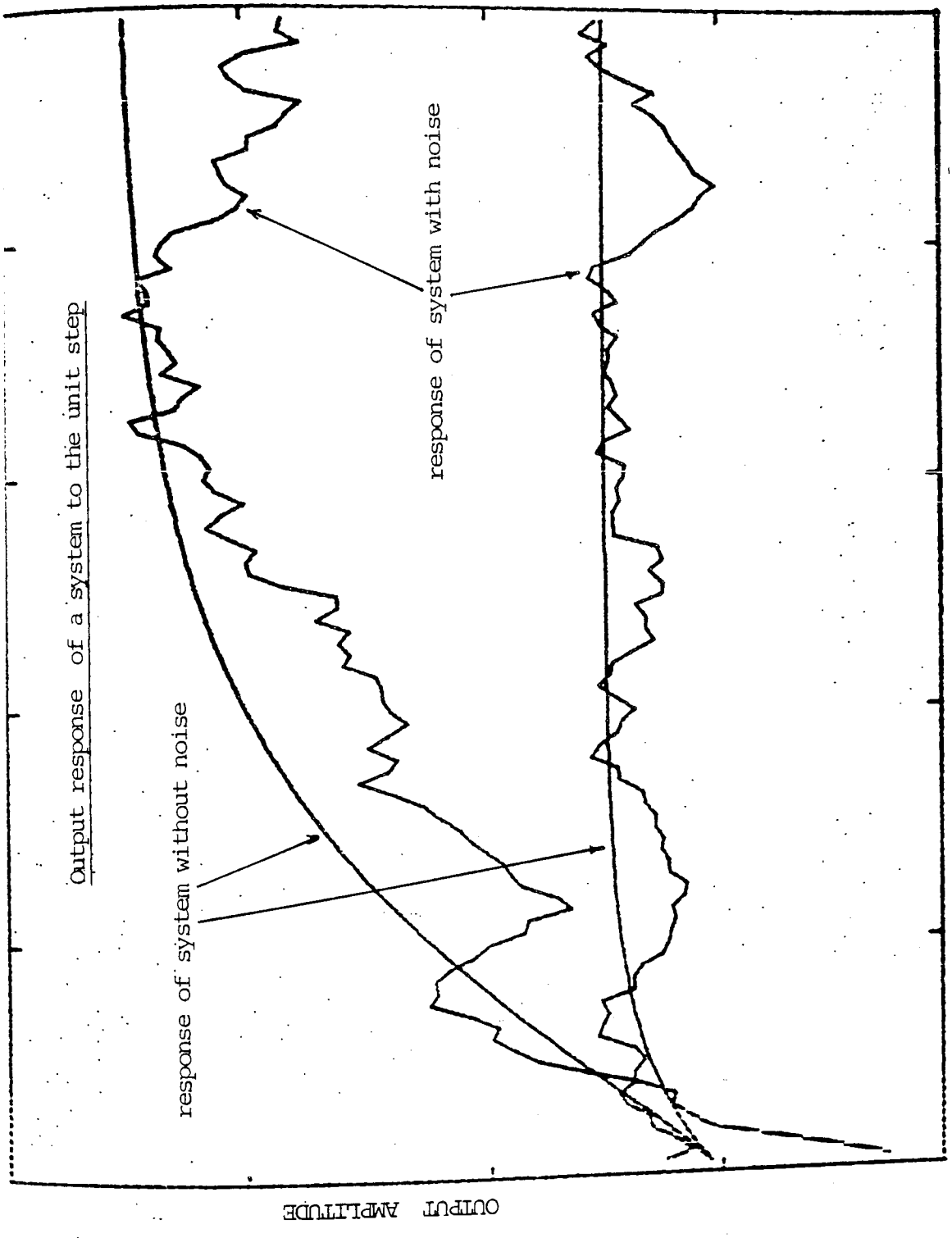


Figure 4.6 (b) State X_{1-2} and $\hat{X}_{1-2}$ against time using standard approach

4.4 COMMENTS AND CONCLUSION

In this chapter the author has endeavoured to show that the observation process can be achieved by using a simpler numerical procedure in the continuous case; it is applicable to the discrete-time case and is substantially faster than that found in the existing literature. This decentralised observation is achieved if each observation process is confined to a sub-system with a structural constraint obtained by decomposing the system into so called **cyclic** and **acyclic** components. It is particularly applicable for a large system where there is a high degree of **sparsity** in the system matrix A.

It should be noted by the reader that a direct comparison was not made with work outlined earlier for the continuous-time case [8,9]. This thesis describes, for the first time, the use of the systematic **partitioning technique** above to make explicit, in the qualitative domain, the **natural processes** of the system. It has been applied to the estimation of a decentralised control system using the technique of Evans et al [7]. Each system used by [8,9] was a **strongly coupled (cyclic)** system which was **decomposed** by imposing certain structure constraints. Their decomposition was achieved neither by identifying the sub-systems explicitly with a systematic decompositon technique nor by observing the dynamics of the system when partitioning into sub-systems. Therefore, the stability of their decentralised observers in a complete system had to be ensured.

From the above discussion, in the continuous case, it is not wise to infer that the other techniques [8,9] are superseded by the approach presented here. These should be used in conjunction with the work presented here if it is not possible to decompose the system or sub-system any further, using the new decomposition technique or by

sensitivity analysis. This structural partitioning technique was also exploited in the discrete-time domain for the first time for the design of decentralised estimators by adopting Kalman filtering.

The theory presented here can be used to handle a non-linear system as well as the time invariant problem, as pointed out by Evans et al [83]; *"the structural analysis applies equally to these problems since connectedness, in the sense of the reachability criteria and term-rank, in the sense of solvability, are purely structural properties which do not take into account the form of connection which can be either numeric or algebraic."* This statement may have a significant bearing on the self tuning adapters used for parameter estimation of a non-linear system.

The identification of the secondary communication necessary to build decentralised observers as a part of distributed control scheme is also developed.

Finally, basic matrix criteria for a **reliable observer** are derived but their solution requires further mathematical development such as can be found in **ref.** [10] and other literature referenced therein. In the next chapter the concepts of reliability and reconfiguration of software processes are presented.

CHAPTER 5

5.0 DESIGN METHODOLOGY

5.1 INTRODUCTION

In a distributed concurrent system the software can be decomposed into a number of processes, the decomposition often being performed on a functional basis [28,90]. Inter-process communications or information flow will take place through defined interfaces to the processes. These information flows are essential to the operation of the complete system. It is essential to limit and control communication under fault conditions to avoid errors propagating through interprocess channels. One way of limiting the extent of information flow is to identify within the collaborating processes those actions which can be grouped as a single task, i.e. treated as an **atomic action** [14].

The flexibility provided by a distributed environment can be exploited to provide dynamic reconfiguration of processes. The survey in section 3.4 showed, that the reconfiguration of processes which service the peripherals, has been ignored in real-time control applications. The general belief was that reconfiguration is rendered ineffective by the inability to move the function of the peripherals. Dynamic reconfiguration must deal with the failure of a process, or an instrument associated with a process, or both.

The aim of this thesis is to develop a methodology for designing distributed computer/microprocessor based Control Systems. Of the four objectives (1) and (2) in the list below were dealt with in chapter 4. The last two objectives (3) and (4) are considered in this

chapter.

(1) Estimation of states.

(2) Systematic identification of communication for distributed control.

(3) Identification of atomic actions.

(4) Dynamic reconfiguration.

The main theme of this chapter is to achieve objective (3) above. It is shown in this chapter that the structural partitions of a linear system, (a mathematical model of a physical plant), can be mapped into the structure of the control software. This forms natural partitions for the software which may be used to identify sections which can be protected by **recovery blocks** or **conversations** [15]. Note here that in the majority of cases the term **structural partitioning** technique implies the decomposition of a linear system into cyclic and acyclic sub-systems but it is also applicable to the further decomposition of a linear sub-system by sensitivity analysis.

This chapter is split into three sections. The first section shows, by considering a specific example, that by decomposition the processes are **partitioned into atomic actions**. This is important when designing distributed systems as it alleviates the construction of fault tolerant systems. The second section deals with the dynamic reconfiguration of specific processes, where the processes are associated with peripherals. The reconfiguration is invoked when either a process fails or the instrumentation associated with it fails. The third section describes the design methodology developed for control of a system in a distributed or decentralised fashion.

5.2 ATOMIC ACTIONS

An atomic action can be defined as follows [12]: *"The activity of a group of components constitutes an atomic action if there are no interactions between that group and the rest of the system for the duration of the activity"*. The **conversation** [15] is a **backward error recovery mechanism** using the idea of **atomic actions** to provide a fault tolerant structure for distributed concurrent systems.

In order to show that the software processes, resulting from the structural partitioning of a linear system, are atomic actions a model representing the **system (software)** must be utilised. In many fields of study, phenomena are not studied directly but indirectly through models. A tool is required to model distributed or centralised systems, which will cope with the interacting processes of the system and allow concurrency to be represented.

The model to be used in this thesis is a **Petri net** [91] description of the system. A formal definition for the basic **Petri net** has been specified [91] together with the **Petri net graph** allowing analysis of the system to be carried out.

Modelling a concurrent system requires a number of additional constructs not required for sequential systems such as **parallelism** and **communications** [92]. These features are incorporated in the concurrent notation **C.S.P.** [93], and concurrent language **Occam** [17]. These additional constructs were modelled using **Petri nets** in Tyrrell's thesis [16], (see appendix D).

Petri nets provide several advantages as a system modelling technique. First, the overall system structure is easy to understand due to the precise and graphical nature of the representation. Second, the behaviour of the system can be analysed using **Petri net** theory and analytical tools [94,95].

It is the aim of this section to show that each partitioned sub-system's control software processes are **atomic actions**. The object is to develop a method for the systematic identification of structural partitions which define **atomic actions**. This identification will be demonstrated using an example defined in chapter 4 which will show the essential features of a control system. Such a system can be modelled using the **SEQ**, **PAR**, **communication** and **asynchronous constructs** available in **Occam**.

5.2.1 Demonstrator Example

Example 2 in section 4.2.1.3 is a linear system with two input variables to control and two output variables to observe it. In example 2 this linear system was partitioned into two decoupled linear sub-systems with one input and one output variable associated with each. These decoupled linear sub-system can therefore be logically mapped into four software concurrent processes for the control software. There are two processes for estimating the state vector of the linear system, and two processes for controlling the state variables of the linear system in a desired fashion. An Occam solution to this is given in **fig.** 5.1. The inter-process communication was derived by applying the primary and secondary communication algorithms.

The control program of **figure** 5.1 can be translated into a **Petri net fig.** 5.2 using the transformation described in Appendix D. This is partitioned by functional attributes into four functional processes which correspond to the actual processes in the program obtained by structural partitioning. The repetitive construct in each functional process gives rise to a loop structure in the **Petri net graph** which serves to bound the graph. The closure of the loops is

signified in **fig.** 5.2 by the primes on the state identifiers (p1', p2', etc.).

By analysing **figure** 5.2, it becomes apparent that after the **coordination** and **synchronisation** communication phase, each functional **process** is **autonomous**. That is **each process** is an **atomic action**: there are no interactions between that process and the rest of the system software processes for the duration of computation. **The implication is that functional boundaries of each process act as a side wall, hence a simple recovery block mechanism may be adequate for each block.**

However in **figure** 5.2 only simple transitions are shown (t_{23}, t_{24}, t_{25} & t_{26}) for the numerical computation (sect 4.2.) of each functional process resulting from the structural partitioning. This complex numerical computation can be parallelised (by the appropriate technique in section 2.2) into concurrent sub-processes, inter-communicating to solve the problem associated with that functional process. Since the communication is not across the functional partition boundaries based on the structure of the physical system these sub-processes can be made **fault tolerant** by using the **conversation mechanism**. The method of conversation placement [16] can be applied.

By identifying these **atomic actions**, made explicit by the structural partitioning technique, it becomes apparent by analysis (see **fig.** 5.2) that the protection against the interprocess communication failure is very necessary if the database is to remain in a consistent state after the update/access mechanism. This fact becomes more acute if the database is distributed when any accesses and updates are performed concurrently. This database is distributed as a result of the structural partitioning of a physical system which naturally decomposes the state vector (run-time, time-critical, database) of a

linear system. Here each disjoint subset of the state vector is associated with a cyclic linear sub-system.

EXAMPLE1.OCC

```
-- program in Occam 1 for controlling a plant
-- Declaration of inter-process channels

CHAN    estrec1, estrec2, actuator1, actuator2, snduvect1,
        snduvect2, rcvstatint1, rcvstatint2, sensor1, sensor2:

-- declaration of process 'control 1'

PROC cont1 (CHAN estrecv, actuator, snduvect) =
  VAR state1, state2, uvect1:
  SEQ
    --
    ... initalise variables
    --
    WHILE TRUE                                          --(t1)
      SEQ
        PAR                                             --(t5)
          --
          -- update process est1 (on channel snduvect) and the
          -- environment (on channel actuator)
          snduvect ! uvect1                             --(t11)
          actuator ! uvect1                             --(t9)
          --
          -- receive the estimated value for state 1 and 2
          -- from process est1 on channel estrecv
          -- note this could have been received in parallel
          -- if wished using two channels
          estrecv ? state1; state2                      --(t10)
          -- END of PAR                                 --(t19)
        -- calculate new trajectory value 'uvect1'
        -- note the process below can be done in parallel
        ... compute trajectory uvect1                   --(t23)
:

PROC cont2 (CHAN estrecv, actuator, snduvect) =
  VAR state3, state4, uvect2:
  SEQ
    --
    ... initalise variables
    --
                                                        --(t4)
    WHILE TRUE
```

```
      SEQ
        PAR                                              --(t8)
          --
          -- update process est2 (on channel snduvect) and the
          -- environment (on channel actuator)
          snduvect ! uvect2                              --(t16)
          actuator ! uvect2                              --(t18)
          --
          -- receive the estimated value for state 3 and 4
          -- from process est2 on channel estrecv
          -- note this could have been received in parallel
          -- if wished using two channels
          estrecv ? state3; state4                       --(t17)
          -- END of PAR                                  --(t22)
        -- calculate new trajectory value 'uvect2'
        -- note the process below can be done in parallel
        ... compute trajectory uvect2                    --(t26)
:

PROC est1(CHAN cntsnd, rcvstatint, sndstatint, cntrcv, sensor)=
  VAR state1, state2, state3, state4, uvect, sensorval:
  SEQ
    --
    ... Initalise variables
    --
    WHILE TRUE                                           --(t2)
      SEQ
        PAR                                              --(t6)
          -- send the estimated value to the process cont1
          -- and to the other estimator ( process est2 )
          --
          cntsnd ! state1; state2                        --(t10)
          sndstatint ! state1; state2                    --(t14)
          --
          -- receive the state variables and control vector
          -- which affect the computation for this estimator.
          -- Also update the reading from the channel sensor.
          --
          rcvstatint ? state3; state4                    --(t13)
          cntrcv ? uvect                                 --(t11)
          sensor ? sensorval                             --(t12)
          -- END of PAR                                  --(t20)
        -- estimate the new value for state 1 and 2
        -- based on the updated values ie state3, etc.
        --
        ... predict values for state1 and state 2        --(t24)
:
```

```
PROC est2(CHAN cntsnd, rcvstatint, sndstatint, cntrcv, sensor)=
  VAR  state1, state2, state3, state4, uvect, sensorval:
  SEQ
    --
    ... Initalise variables
    --
    WHILE TRUE                                          --(t3)
      SEQ
        PAR                                             --(t7)
          -- send the estimated value to the process cont2
          -- and to the other estimator  ( process est1 )
          --
          cntsnd ! state3; state4                       --(t17)
          sndstatint ! state3; state4                   --(t13)
          --
          -- receive the state variables and control vector
          -- which affect the computation for this estimator.
          -- Also update the reading from the channel sensor.
          --
          rcvstatint ? state1; state2                   --(t14)
          cntrcv ? uvect                                --(t16)
          sensor ? sensorval                            --(t15)
          -- END of PAR                                 --(t21)
        -- estimate the new value for state 3 and 4
        -- based on the updated values ie state1, etc.
        --
        ... predict values for state3 and state 4       --(t25)
:


--
-- main program
--
PAR
  cont1(estrec1, actuator1, snduvect1)
  cont2(estrec2, actuator2, snduvect2)
  est1(estrec1, rcvstatint1, rcvstatint2, snduvect1, sensor1)
  est2(estrec2, rcvstatint2, rcvstatint1, snduvect2, sensor2)
```

FIGURE 5.1 Occam Program for a subset of a Gust Alleviation Control System

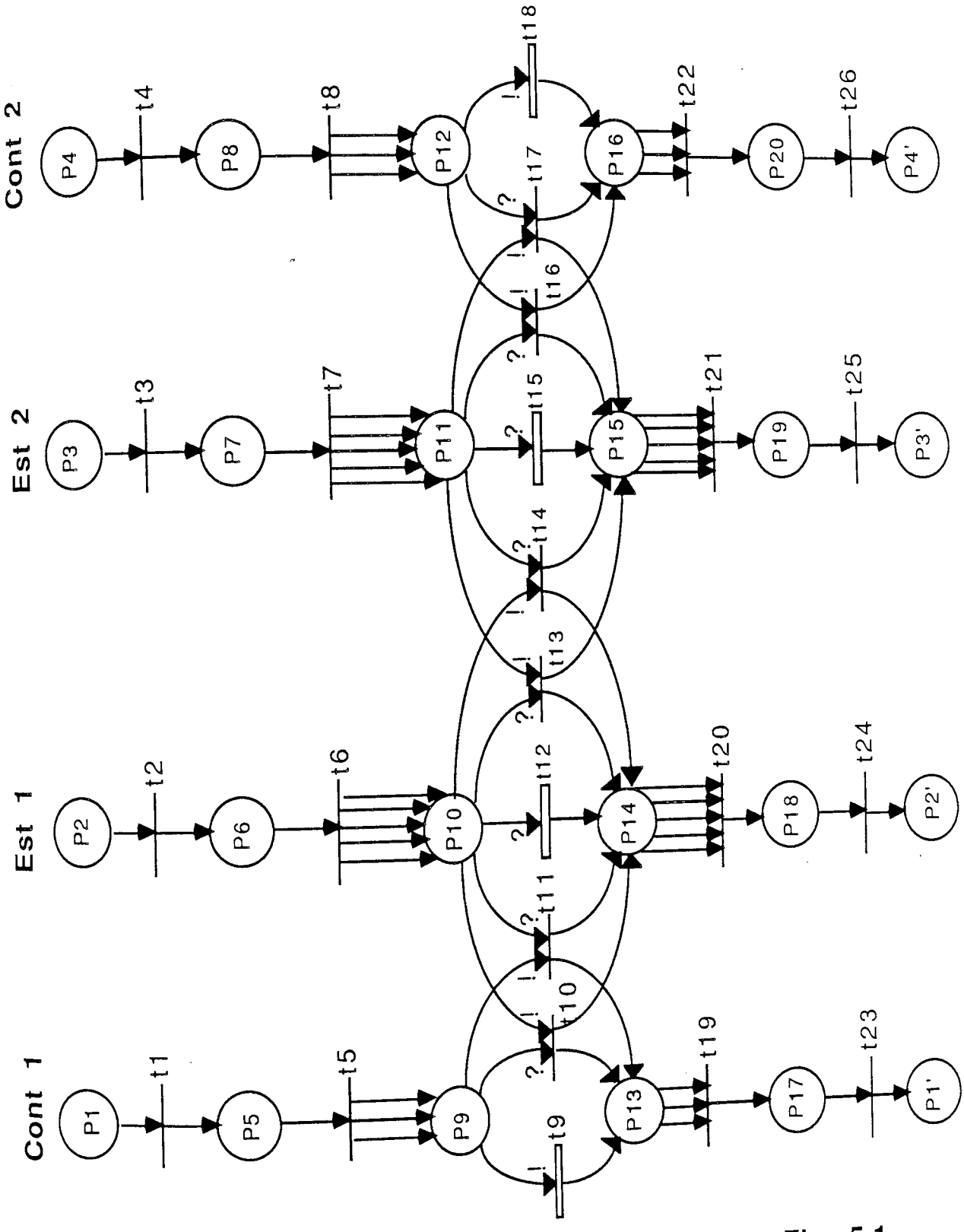


Figure 5.2 Petri Net of Occam Program in Fig. 5.1

The **integrity** of inter-process **communication** was investigated in [16]. It can be shown that the basic communication primitives used in message passing systems have deficiencies when applied to systems with safety implications [18]. It has been proposed that the integrity of a system, which involves inter-process communications, may be increased by introducing a **time-out mechanism** to avoid **deadlocks**. This **time-out mechanism** will guarantee that each **atomic action** will continue (or at least terminate). **Petri net** models were used to identify a boundary for a **time-out mechanism** [16]. However, this thesis did not address the problem of **protecting a distributed database**. Only a preliminary survey was carried out in this thesis; it requires further research to identify an appropriate protection mechanism. Section 5.2.2 shows some of the problems associated with it and gives an example of a technique used in protecting the database.

5.2.2 Reliable Distributed Database

A distributed database provides certain potential advantages relative to a centralised database. One of the advantages is that it can process requests in parallel. This parallelism, through proper design, can be exploited to provide more redundancy and hence, higher reliability, compared to the centralised system. However, several problems are seen that either do not exist or are less likely to occur in a centralised system, such as communication integrity for transactions. Formally, if a procedure accesses and modifies the database in such a fashion that the data is consistent before the transaction is executed, then the data will also be consistent after the transaction is completed.

A real-time database is considered to be an important part of any process control system, especially if it is distributed. A real-time database will be an integral part of the Distributed Computer Control System (**DCCS**) and will consist of [96]:

- the process control database
- the management database for **DCCS**
- the program library (software bank)

The database availability is an important requirement in a distributed computing environment for a real-time, time-critical system. This can be improved by addressing problems such as **Reliability**, **Integrity**, **Security** and **Responsiveness**. In providing **fault tolerance**, it is necessary to ensure that the consistency of process control database is not upset. An insight into database requirements can be found in [96].

When the real-time database is to be distributed the designer must consider how to decompose the real-time database, how to design the communications network to support interprocess message flow between the components of the database and how to control the **concurrent transactions** so the database remains consistent.

The database may be split according to geographical or functional considerations or to provide redundancy for reliability. The **partitioning technique** discussed earlier in the thesis (section 2.2.3), i.e. decomposition based on the physical structure of the plant, **dictates the structure of database and communication**. The structural partitioning technique decomposes the state vector (real-time database) of a linear system into disjoint subsets of the state vector which are associated with every cyclic linear sub-system. Therefore communication is required for a decentralised control scheme as pointed out in sect. 3.1.

There are other decomposition techniques based on the functional relationship between the attributes of a data set. These

techniques are summarised in [97]. The authors of that paper show that functional relations can be restated using Boolean algebra, allowing the designer to explore the properties of a given set of functional relations, as well as in the task of partitioning a data set into sub-files for efficient implementation.

Consistency in the stored data must be maintained despite failures and without restricting the concurrent processing of application requests unnecessarily. In the database literature, a **transaction** [98] is defined as an arbitrary collection of operations bracketed by two markers: **Begin Transaction** and **End Transaction**, which has the following properties:

(i) **Failure Atomicity**: either all or none of a transaction is performed.

(ii) **Permanence**: if a transaction completes successfully, the results of its operations will never be lost subsequently.

(iii) **Serializability**: if several transactions execute concurrently, they affect the database as if they were executed serially in some order.

(iv) An incomplete transaction must not reveal results to other transactions, in order to prevent Cascading Aborts if the incomplete transaction has to be undone subsequently.

Transactions simplify the treatment of failures and concurrency. **Failure atomicity** makes certain that when a transaction is interrupted by a failure, its partial results can be undone. **Serializability** ensures that other concurrent transactions cannot observe these inconsistencies. **Preventation of cascading aborts** limits the extent of error recovery.

Each transaction must start with a **BEGIN** operation and terminate with either **COMMIT** or **ABORT(RECOVER)**, i.e. **two phase commit protocol**. These primitives will need to be supported in a recovery mechanism. These primitives are supported by the

International Standard Organisation (ISO) Commitment, Concurrency and Recovery (CCR) protocol specification [99]. A recovery point is established by **BEGIN**, and is subsequently discarded by **COMMIT** or restored by **ABORT**. This may be suitable for the design of a **fault tolerance mechanism**.

5.3 DYNAMIC RECONFIGURATION

The literature survey (sect. 3.4) showed that little consideration has been given to the dynamical reconfiguration of processes in real-time control system. Two examples are presented in this thesis to demonstrate that dynamic reconfiguration is possible for the processes that are associated with the actuators and sensors.

Dynamic reconfiguration will normally place a severe demand on the system software, on top of the demands for high level performance in a computational system which is monitoring and controlling a plant [100]. For example, the application may involve real-world data acquisition, complex arithmetic calculations which perform the state vector estimation, generate control outputs to the application plant and system level tasks to control and schedule computational objects.

If each software **process**, extracted via the **structural partitioning**, is an **atomic action**, the process can be made resilient using the conversation mechanism. Protection against design faults is provided by offering a set of alternate blocks, each employing a different algorithm to achieve the same objective. If the primary fails, then an alternative is tried. Since **dynamic reconfiguration** will take place at the **partition boundary** using the technique in this thesis, this reconfigurable process will be incorporated in an alternate block. This can be regarded as an anticipated fault rather than the unanticipated faults normally catered by the recovery mechanism.

Note: when the environment demands a fail safe design, the time ovehead required to reconfigure must be accounted for such that a real-time system must be able to respond to an external stimulus within a specific period known as the **critical time**. Atomic actions are relocated so that the dynamics of the linear system do not become

unstable or the problem of synchronisation and coordination does not occur as highlighted in chapter 1.

Two examples are used to demonstrate the possibility of dynamic reconfiguration. Quantitative values are not necessary to show dynamic reconfiguration. No assumptions are made here as to whether the computing element is a single or multiple processor system nor whether it uses centralised or decentralised control; these issues are not important at the present.

Example 1

The graphical form of the state space equation of the second example in section 4.2.1.3 is shown in **figure** 5.3 in digraph form in the Boolean domain.

It can be seen from the digraph that from control variable U_1 or U_2 there is a potential controllabilty of state variables 1, 2, 3, and 4. From the output variables Y_1 and Y_2 there is a potential observability of state variables 1, 2, 3, and 4. It was shown in section 4.2.1.3 that this particular linear system can be **decomposed** into two sub-systems, which are **dynamically decoupled**: one sub-system containing state variables 1 and 2 and the other 3 and 4. The linear system, being decoupled, can be controlled and observed by four concurrent processes as described in section 5.2 and these are shown graphically in **fig.** 5.4. The point is that either of the observation processes can estimate the state vector of the complete linear system, and either of the control processes has the ability to control the trajectory of controlled plant.

The dynamic reconfiguration of processes in this system is shown in **fig.** 5.5 which indicates which state variables each process

is catering for. The four figures indicate how the processes reconfigure when one of the process fails either due to a (programming) design fault or a fault in the interface to the environment. Other possibilities, such as a combination of two processes failing can also be catered for here except for the complete failure of either of the two control processes or the two estimation processes at the same time.

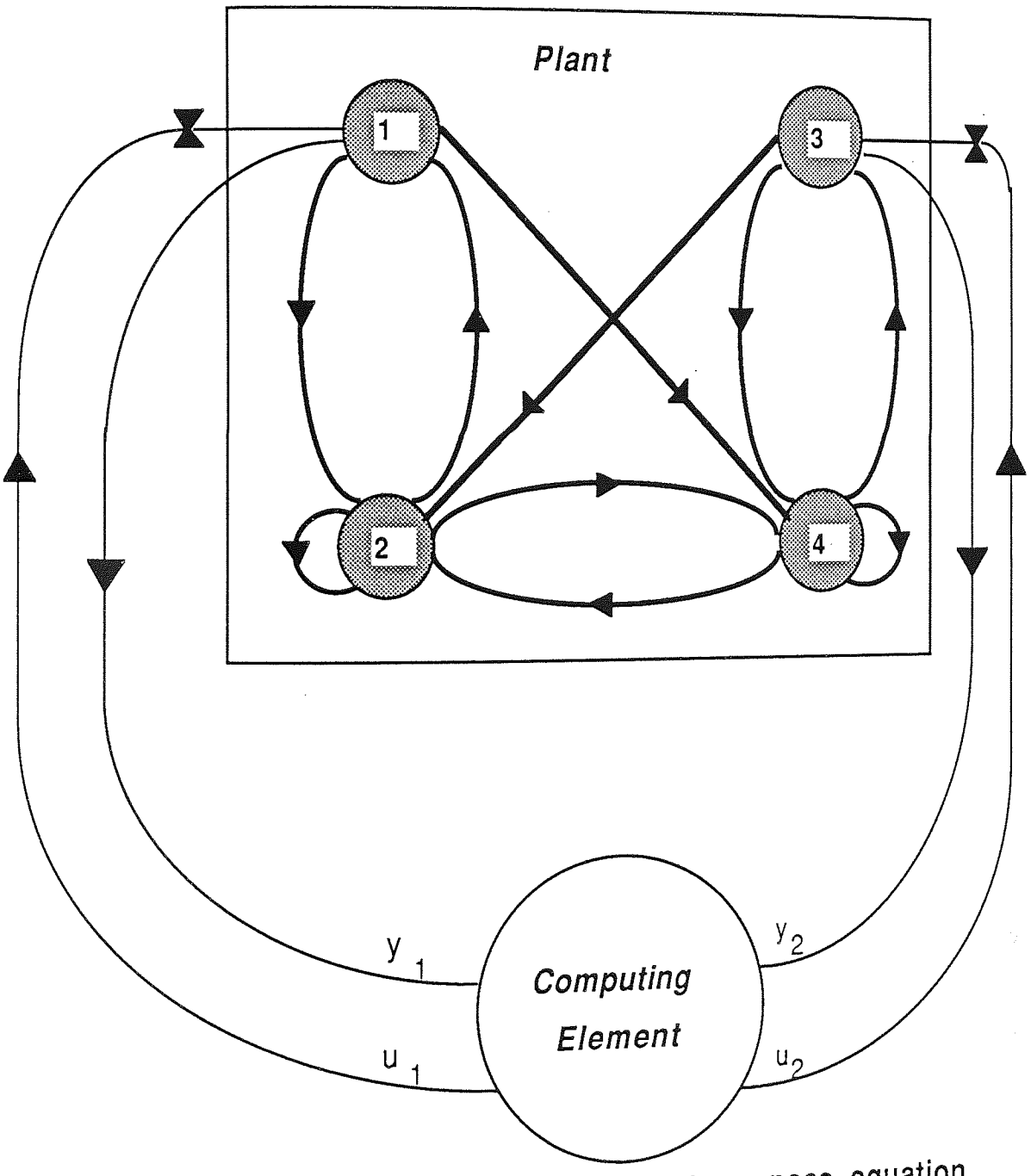


Figure 5.3 Graphical representation of state space equation of Gust Alleviation Control sub-problem

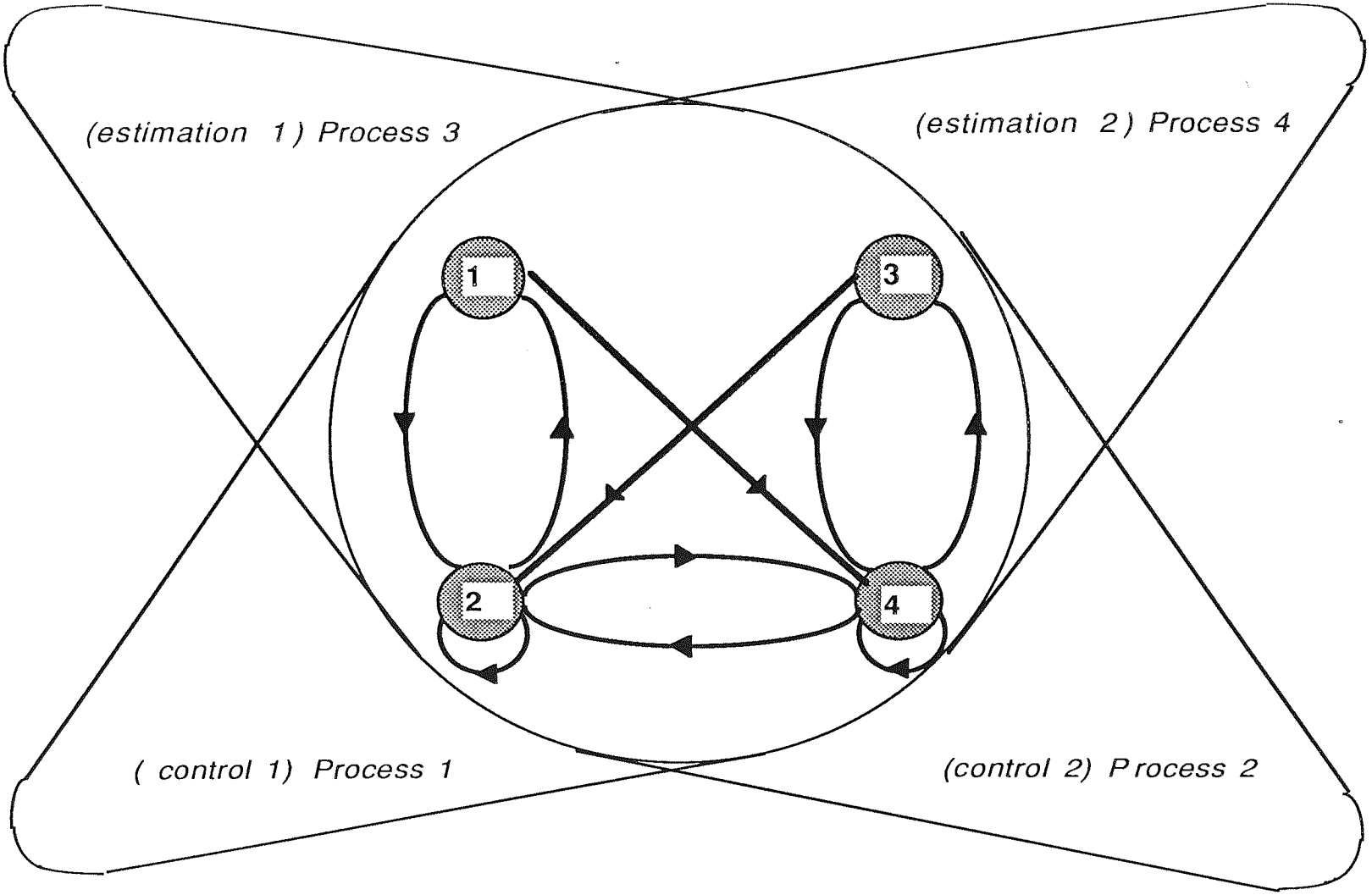


Figure 5.4 (a) Potential Controllable and Observable space

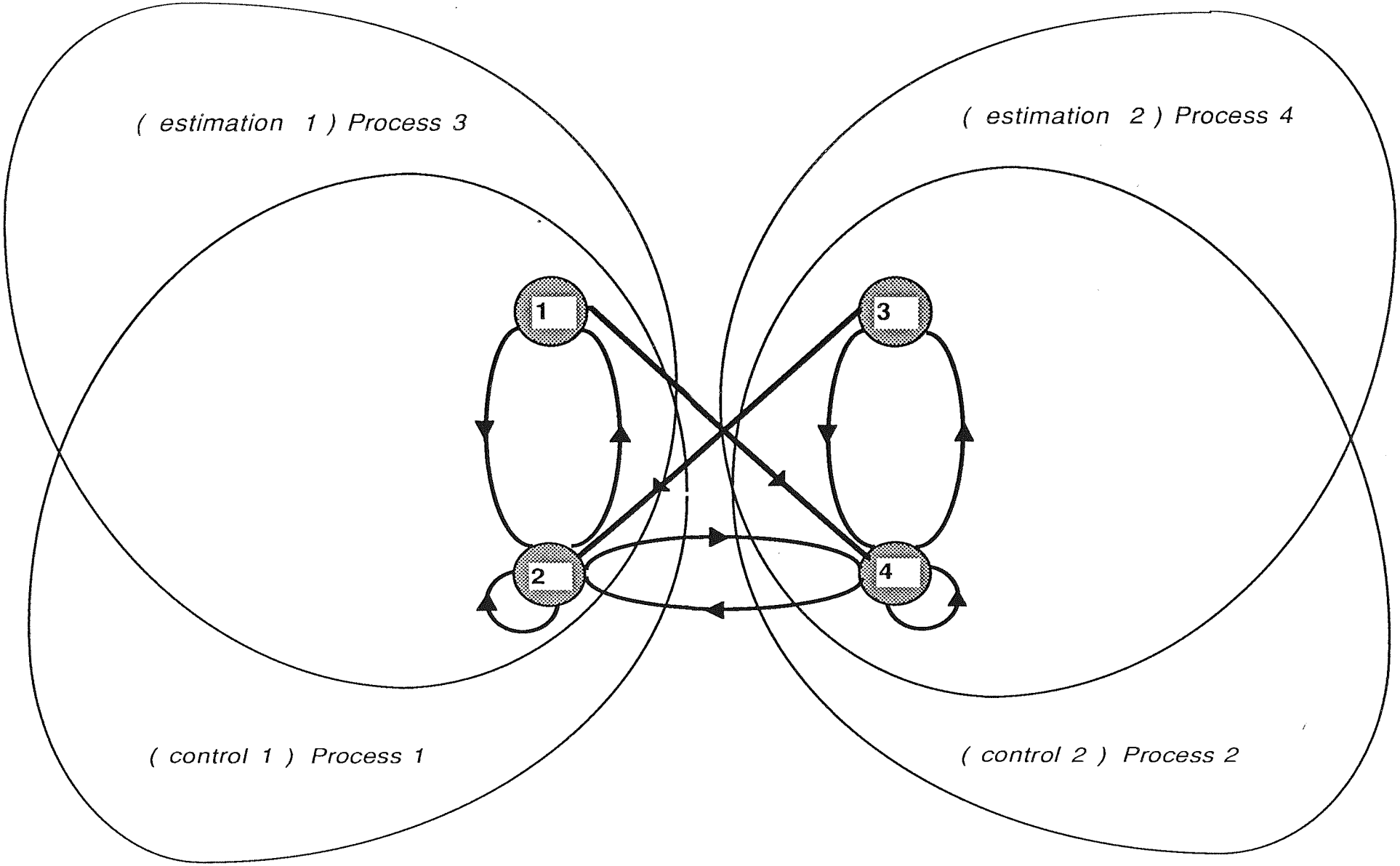


Figure 5.4. (b) Designated controllable and observable space

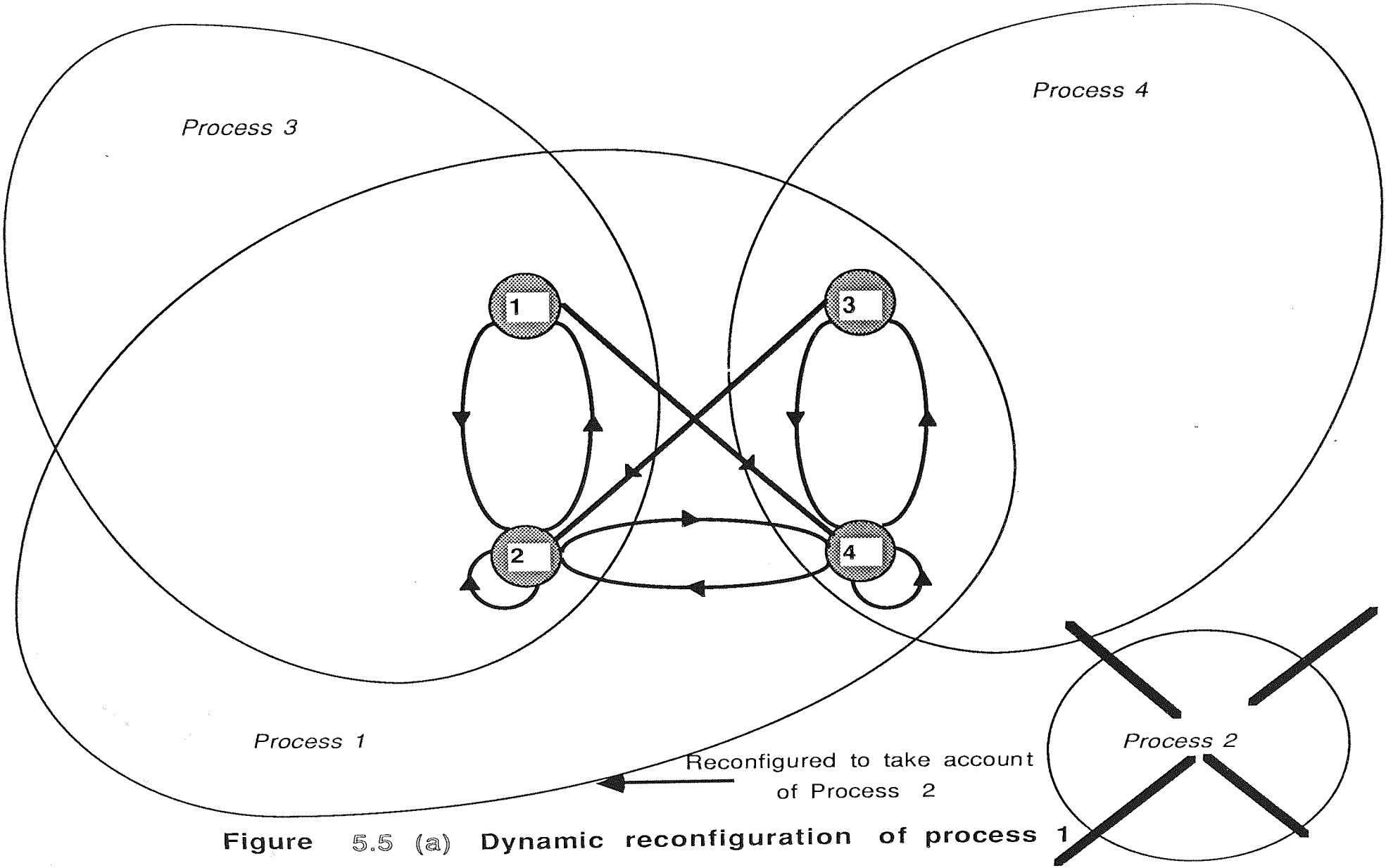


Figure 5.5 (a) Dynamic reconfiguration of process 1

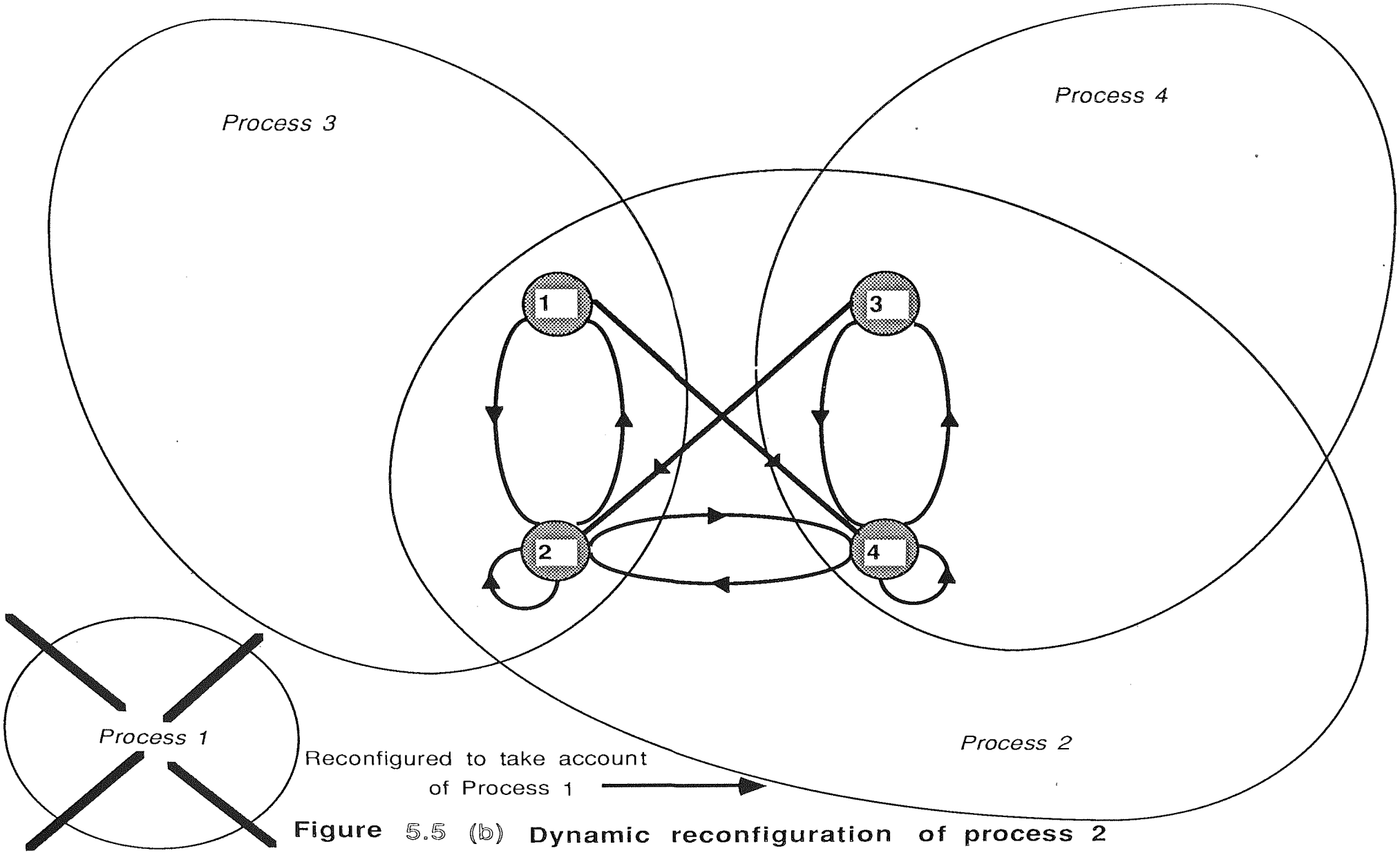


Figure 5.5 (b) Dynamic reconfiguration of process 2

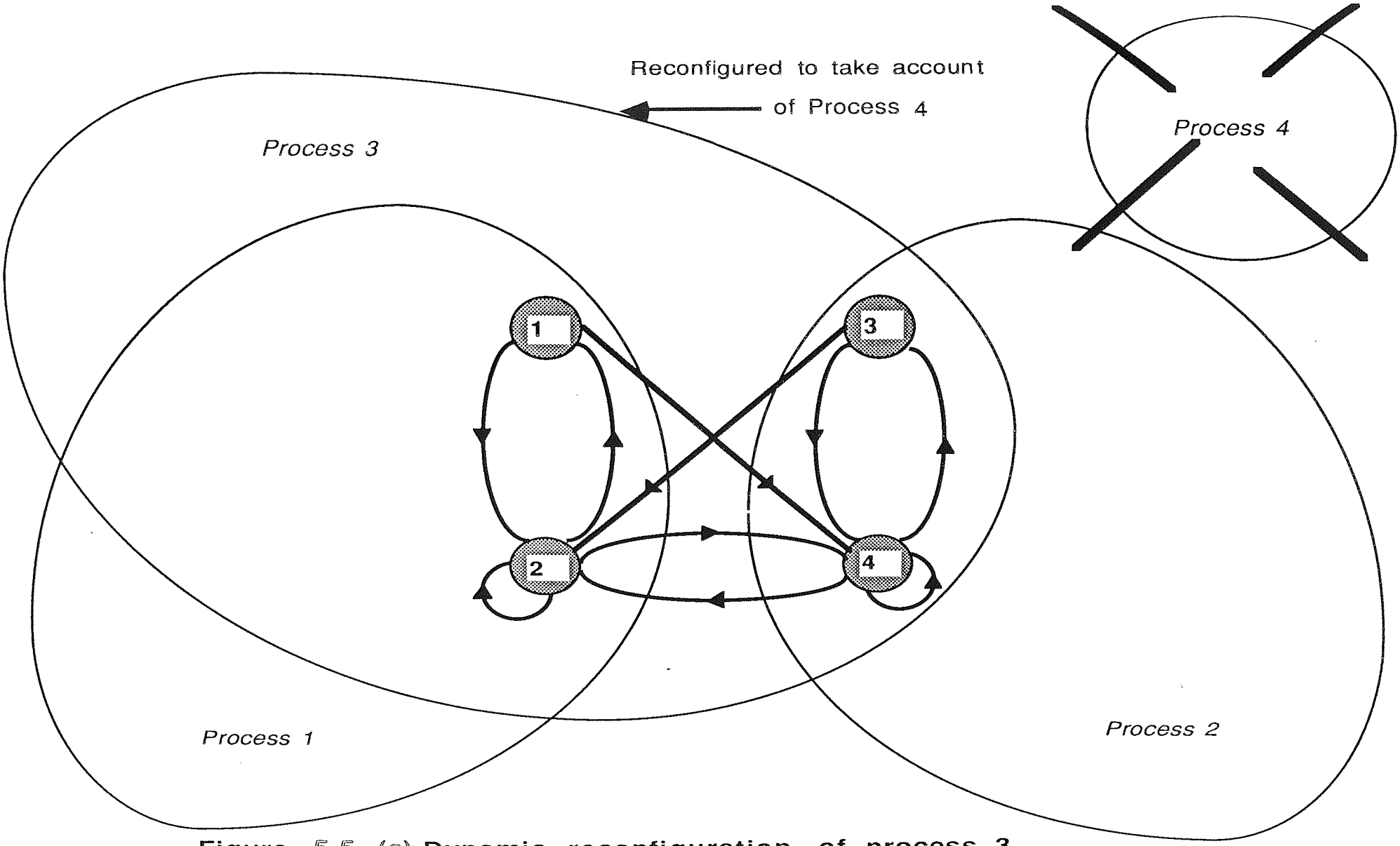


Figure 5.5 (c) Dynamic reconfiguration of process 3

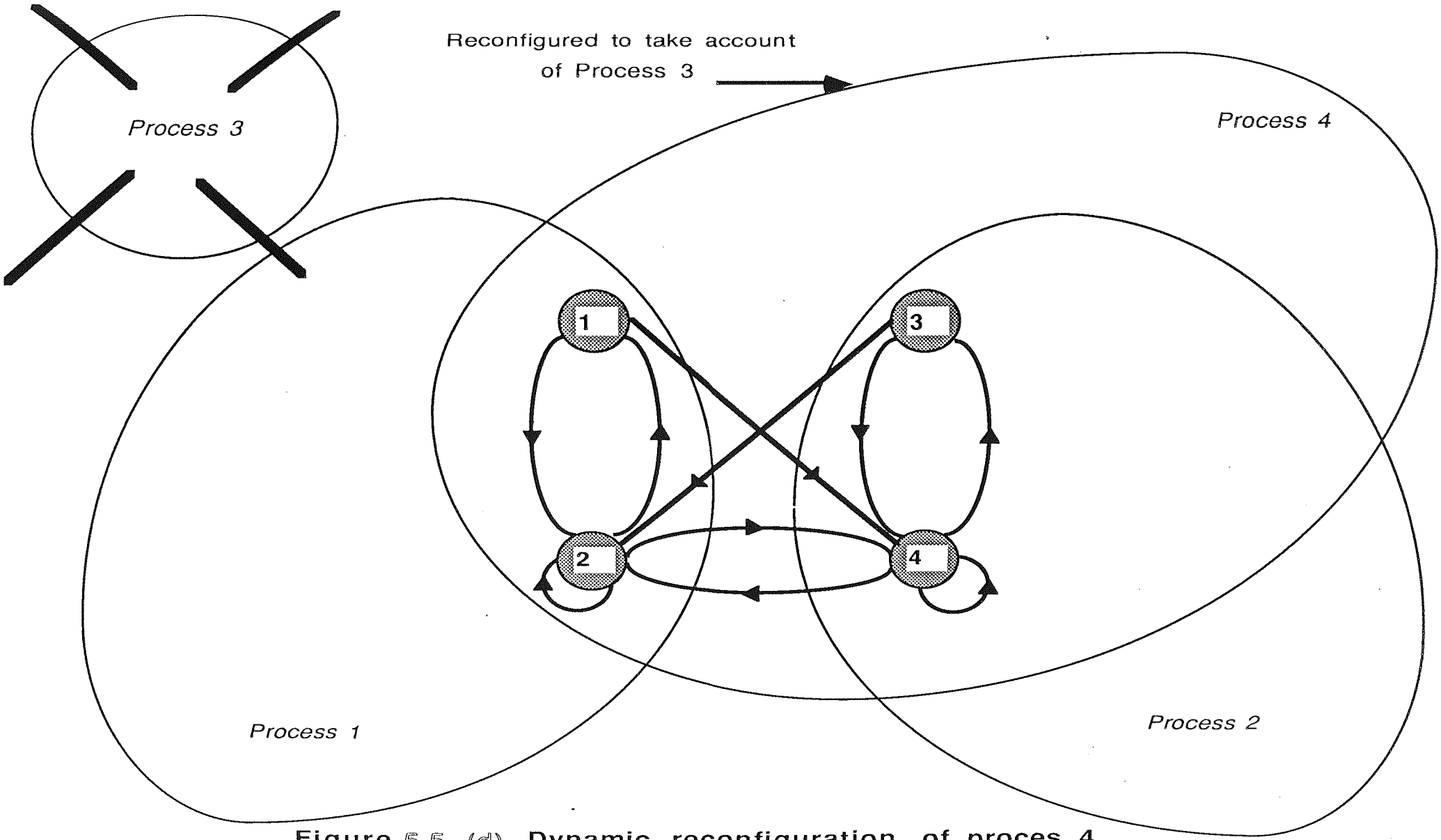


Figure 5.5 (d) Dynamic reconfiguration of proces 4

Example 2

The state space equation of the first example in section 4.2.1.3 is shown in **figure** 5.6 in a digraph form in the Boolean domain.

It can be seen from the digraph that the control variable U_1 has the potential controllability of state variables 1, 2, 3 and 4. The output variable Y_2 has the potential observability of state variable 1 and 2 only, whereas the variable Y_1 has the potential observability of all the state variables. It was shown in section 4.2.1.3 that this system can be decomposed into two **cyclic** linear sub-systems. Due to the system being decoupled, this linear system can be controlled and observed by three concurrent processes and not four. These are shown graphically in **fig.** 5.7. One of the observation processes can estimate the state vector of the complete linear system, and the control process has the ability to control the trajectory of the controlled plant.

The dynamic reconfiguration of processes in this linear system is shown in **fig.** 5.8 with an indication of the state variables which each process caters for. The figures indicate how the observation process reconfigures when the other observation process fails either due to a (programming) design fault or to a fault in the interface to the environment. Two points can be made, the second of which is very significant.

(i) Reconfiguration is asymmetrical

(ii) It indicates how the system could be made resilient against failure.

The interface to the environment provides protection of observation process1 and its sensor but the failure of either U_1 and Y_1 and the associate processes will be catastrophic. Therefore, analysis of the system for dynamic reconfiguration provides, as a by product, identification of the potential weaknesses of the controller that is,

whether the external interface or the computing element should be duplicated to provide the fault tolerant structure.

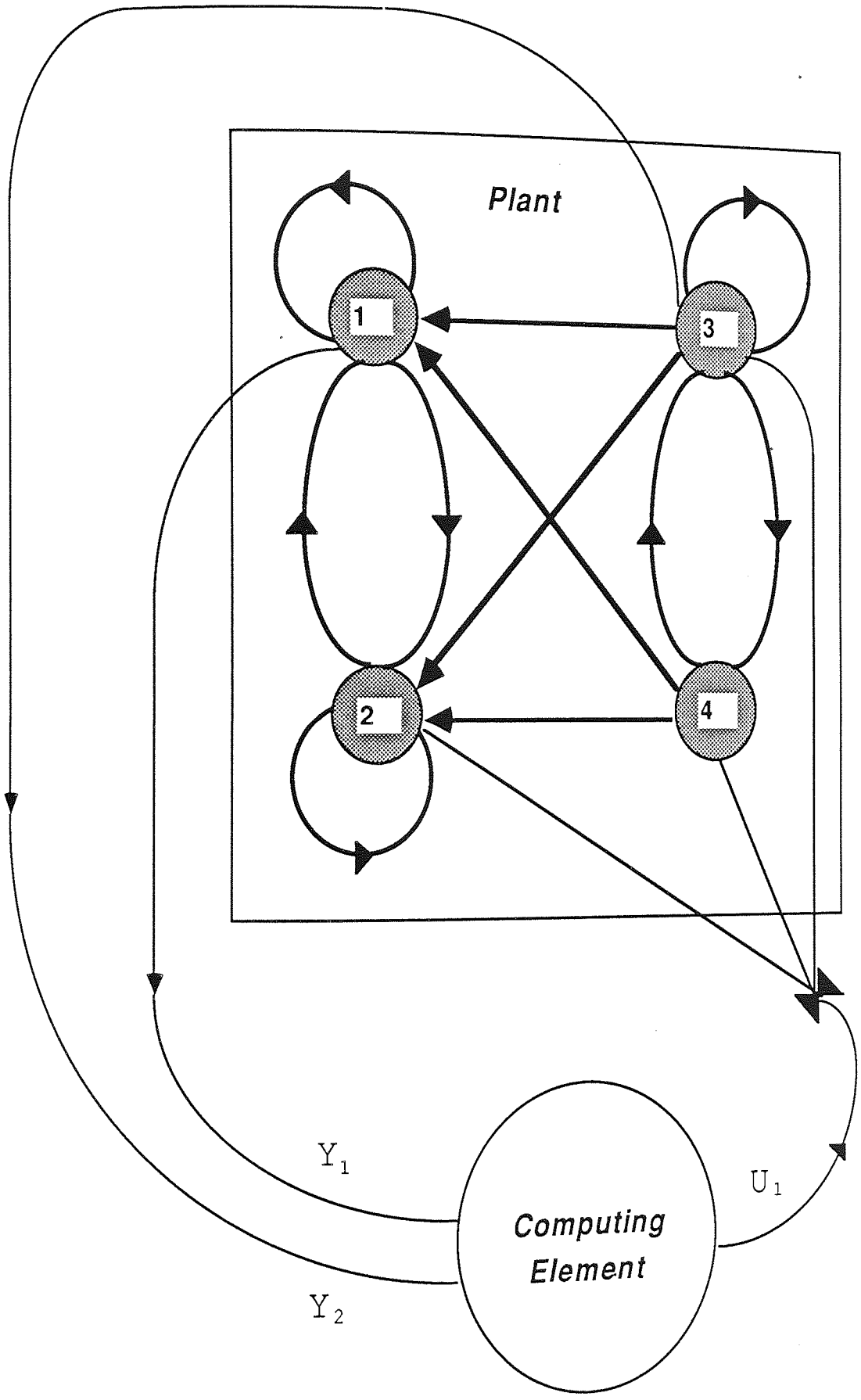


Figure 5.6 Graphical representation of state space equation of a control problem

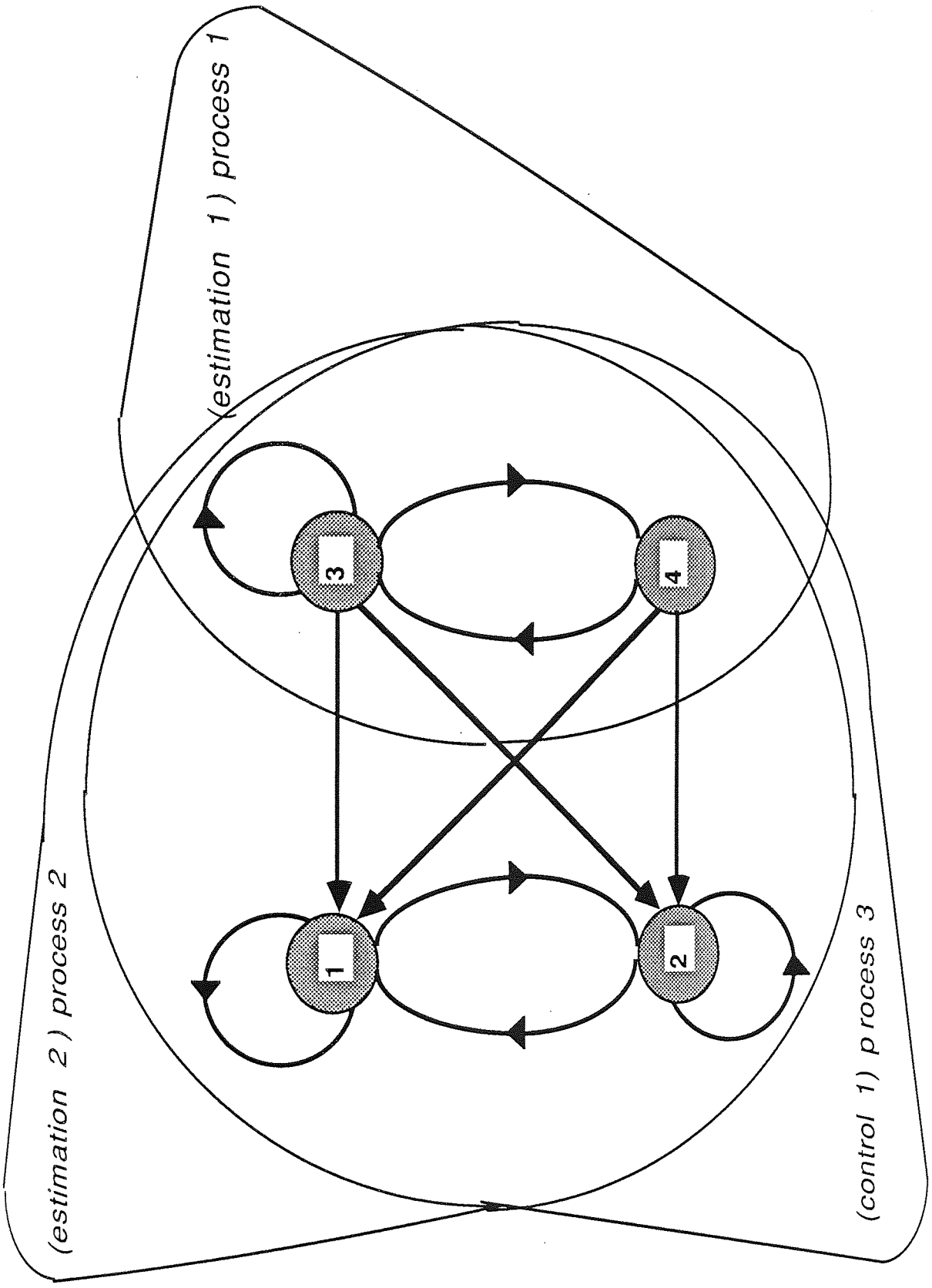


Figure 5.7 (a) **Potential controllable and observable space**

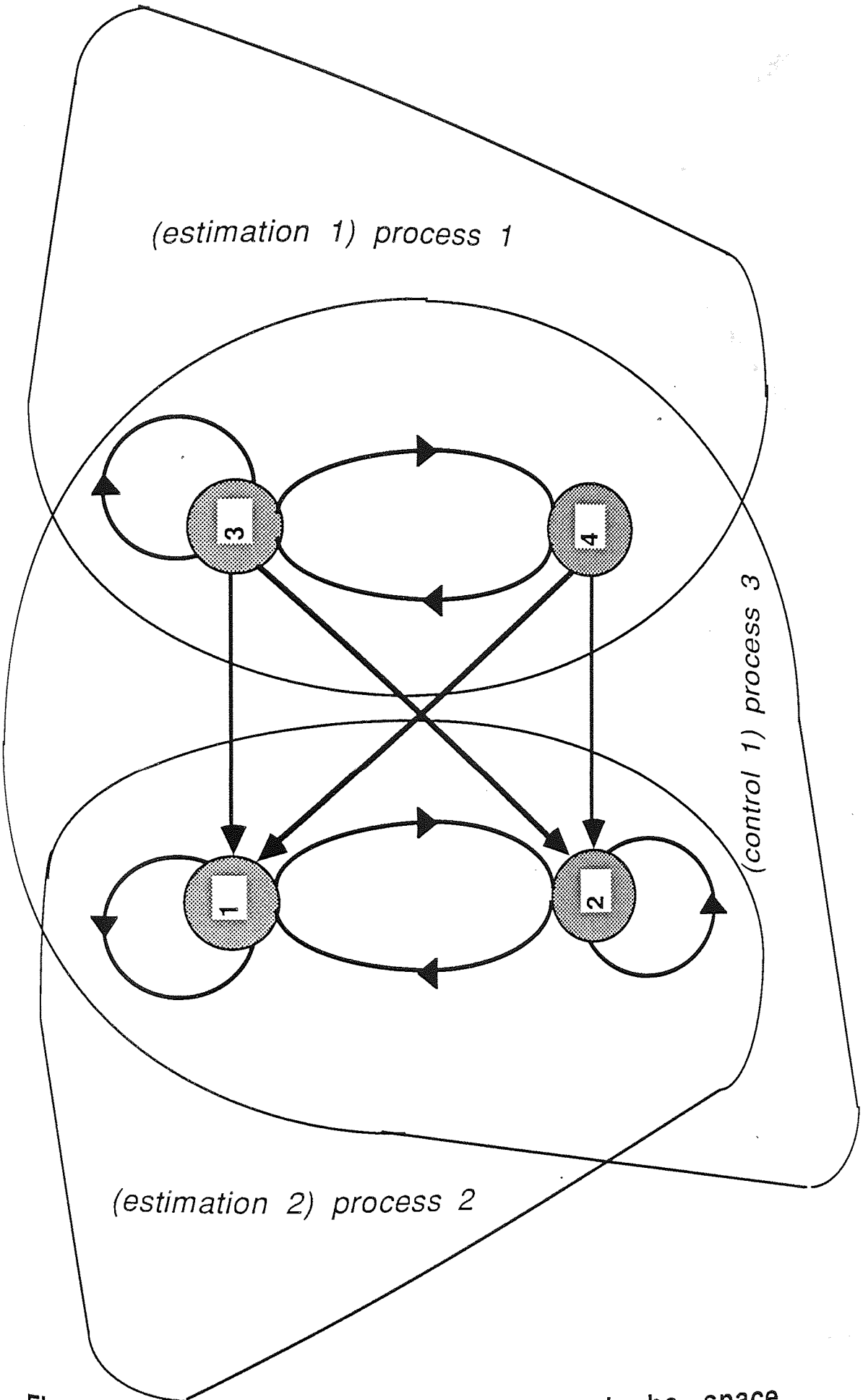


Figure 5.7 (b) **Designated cont. and obs. space**

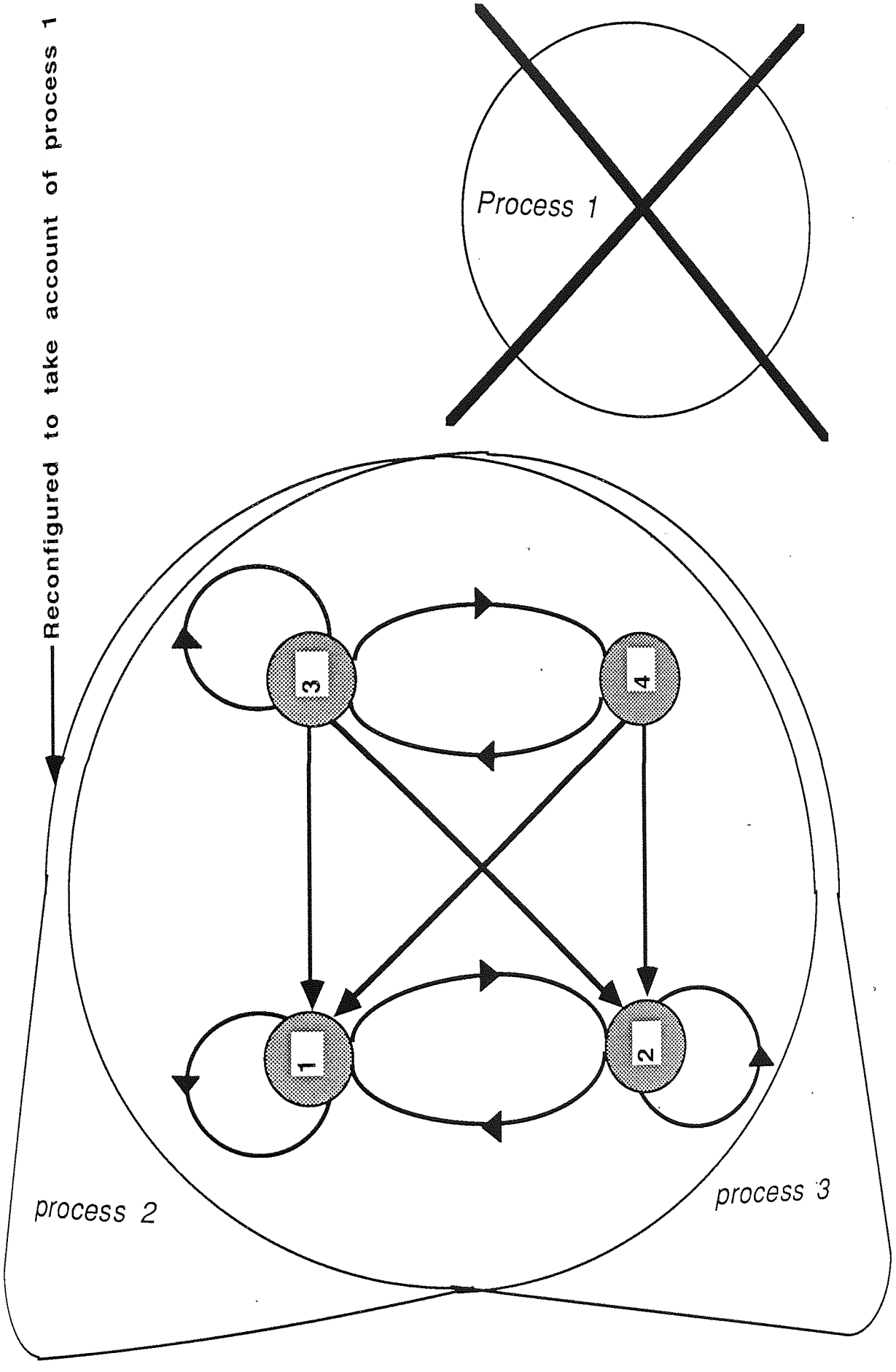


Figure 5.8 Dynamic reconfiguration of process 2

5.4 DESIGN METHODOLOGY

The methodology put forward in this thesis for the design of Distributed Microprocessor based Control Systems, and the sequence of activities needed to be performed, is shown in schematic form in **figure** 5.9. The * on the diagram indicates where new work was carried out in the research. These areas were elaborated in chapter 4 and in this chapter.

The design methodology put forward is structured completely around the **partitioning technique** described in section 2.2.3. This technique makes explicit the natural concurrency of the plant to be controlled. This **partitioning technique** was used by Evans et al [7,83] for decentralised control of a plant by achieving the desired eigenvalues for stability.

The design methodology involves 5 five major activities: **Partitioning**, **Dynamic Reconfiguration**, Identifying **Atomic actions**, identifying **Necessary Communications** and **Optimal allocations**. These activities are summarised now:

5.4.1 Partitioning for Observation and Control processes

The partitioning technique is used to make explicit the natural concurrent processes of a physical system for the control software. The extraction of concurrent processes from the plant represented by the state space equation (**equ.** 2.1) is achieved by using the first four design procedures in [7]. These are as follows:

(a) Form Boolean matrix A_b.

(b) Generate reachability matrix R

(c) Generate permutation matrix P

(d) Decompose system by transformation P^tAP.

Once this physical extraction has taken place (step d), it is necessary to see whether these physical processes can be exploited to achieve a maximum number of logical (computational) processes associated with the linear sub-system for controlling purposes. This maximum number is always equal to twice the number of physical processes that can be extracted. The factor of two is possible because a duality exists between the linear system being controllable and the linear system being observable, as defined by the input and output interface to the system.

For any particular linear system there is a unique set of system inputs and outputs. These are physical impositions and for a real system can only be altered by making alterations to the physical system [80]. Therefore, the maximum number of logical processes can never be achieved in practice (see example 2 in section 5.3). However, the output sensors can normally be positioned anywhere on the system as required, if it is to be cost effective or manufacturable. Hence, there will be (in the majority of systems) a one-to-one correspondence between logical and physical processes for the estimation of the state vector of the physical system.

5.4.2 Dynamic reconfiguration

The potential dynamic reconfiguration is depicted using the techniques (by use of the graphical and the potential controllability and observability technique) as shown in section 5.3. The identification of the dynamic reconfiguration of logical processes will also indicate the potential weakness of the system to certain types of failures. This may also indicate where to introduce fault tolerance into hardware and software.

The steps are as follows:

(a) Determine the entire reachable sub-space for each input and

output variable.

i.e. Ru_i and Ry_i $\quad i = 1,2,\ldots,s$

(b) Determine the compatible controllable and observable sub-spaces for each input and output variables, see section 3.2.

i.e. ${}^{j}K_i$ and ${}^{j}M_i$.

(c) Use the decomposition technique of section 5.4.1, (i.e. extracting natural concurrency), so that each logical process associated with the environment interface controls or observes a designated subset of state variables from its complete set i.e. K_i and M_i respectively. This will lead to an increase in the speed, and reliability of the process, etc.

(d) Then check, graphically or using set theory, if other processes can control or observe these state variables if the prime process fails or the associated interface malfunctions. The necessary information to reconfigure is contained in the K_i and M_i vectors. Also ensure that on reconfiguration the partition boundaries are not violated.

5.4.3 Fault Tolerant Structure

The identification of atomic actions is made explicit by the partitioning technique. Therefore at the outset only the recovery block technique is necessary for each partition. One of the alternate blocks will be for the dynamic reconfiguration process if it is possible to reconfigure.

Note: the steps are identical to those achieved in the previous sub-section (5.4.2), i.e. in dynamic reconfiguration. It is necessary for dynamic reconfiguration to be considered first so that it can be incorporated in the recovery block mechanism.

The control software processes extracted by structural

partitioning are composed of asynchronous communicating sub-processes (see chapter 4 for estimation process equations). If these sub-processes can be executed in parallel, the conversation mechanism is required if a **domino effect** is to be avoided. Within the partition boundary, the atomic actions cannot be identified at one level higher than the coding stage. Therefore, to achieve fault tolerance for these sub-processes, the method of conversation placement can be applied [16].

The placement method is applied as follows.

(1) code the algorithm in concurrent language i.e. Occam.

(2) represent the code in a Petri net model

(3) derive the reachability tree from the Petri net model

(4) analyse the reachability tree, from one state to another and then derive the state change table caused by communications.

(5) identify the conversation block by first deciding where the recovery point and acceptance test is to be placed. Then the set of entry and exit processes are identified by set theory.

Note that nested conversations can be identified using the technique in [16]. However, it is important to realise that the state change table is used for a branch of the tree only, that is, the placement method is applicable only for each branch of the reachability tree as pointed out in **ref.** [16].

5.4.4 Identifying Necessary Communications

The role of communication in the control of distributed processes was emphasised in chapter 3. It is necessary to identify the necessary communication in order to make the system controllable in a decentralised fashion. Communications are necessary because the database of the controlled system is partitioned.

The identification of **secondary (extended) communication** was shown in section 4.3. This may be amalgamated with the identification of necessary communication, as described in section 3.2. Thus:

(a) Find the compatible reachable sub-space of each input and output variable.

i.e. Ru_i and Ry_i $\quad i = 1,2,\ldots,s$

(b) Find the compatible controllable and observable sub-space for each input and output variable, (section 3.2.)

i.e. K_i and M_i.

(c) Find the common sub-space for each station (computational object). Here each station requires a hardware resource but not necessarily one computing element per station. Then these objects may be used in optimal allocation procedure.

i.e $L_i = K_i \cap M_i^t$

(d) Check if communication is required for system to be controllable in a decentralised structure.

$$\bigcup_{i=1}^{s} L_i = \text{Unit vector}$$

If the equality holds then go to step (g) otherwise continue sequentially.

(e) Find the necessary communication for each station i

(f) Update the common sub-space arising from communication

(g) Designate the sub-space which the station should be controlling and observing. This should be dictated by the structural partitioning.

(h) For the system to be observed, that is for estimating the values for state variables, find the necessary, (secondary), communications. These are the interactions from other stations.

Note: steps (e) and (h) define the communication structure required for the system to be controlled in a decentralised fashion.

5.4.5 Optimal allocation

In the previous section it was emphasised that each station will require a hardware resource. In the final development process, eventually, there will be a requirement to match the logical resources to the available computing elements, such as processors/computers.

Optimal allocation is the allocation of the **computational objects**: (processes, databases, etc.), so that the actual cost of running the computation and the inter-processor communication is minimised. This can be achieved by combining the **PROXCUT** algorithm, proposed by Jenny et al [40] and the model used at Brown University by Stone et al [41], (which were described fully in section 2.2.2). The work by Jenny et al minimised the Interprocessor Communication Overhead involved; the work by Stone et al also included the minimisation of the cost of running the computation as well. However, while the algorithm of the former is superior, the latter modelling technique was better.

The above techniques concern the static assignment of computational objects to the available hardware resources. Optimum dynamic assignment of the general purpose processes is not addressed in the design methodology proposed in the thesis. However, the work by Bokhari [45] describes how the dynamic assignment of a modular program can be achieved optimally. The model used by Stone is extended to include the cost of reassigning computational objects from one processor to another dynamically and the cost of that object residing in the hardware without being executed. This technique would put a severe

demand on the run-time system software required to monitor the activity of objects and running the above algorithm in the background. How often this reassignment takes place in real-time, will dictate whether or not executing/invoking this task would lead to instability in the **Operating System Software**.

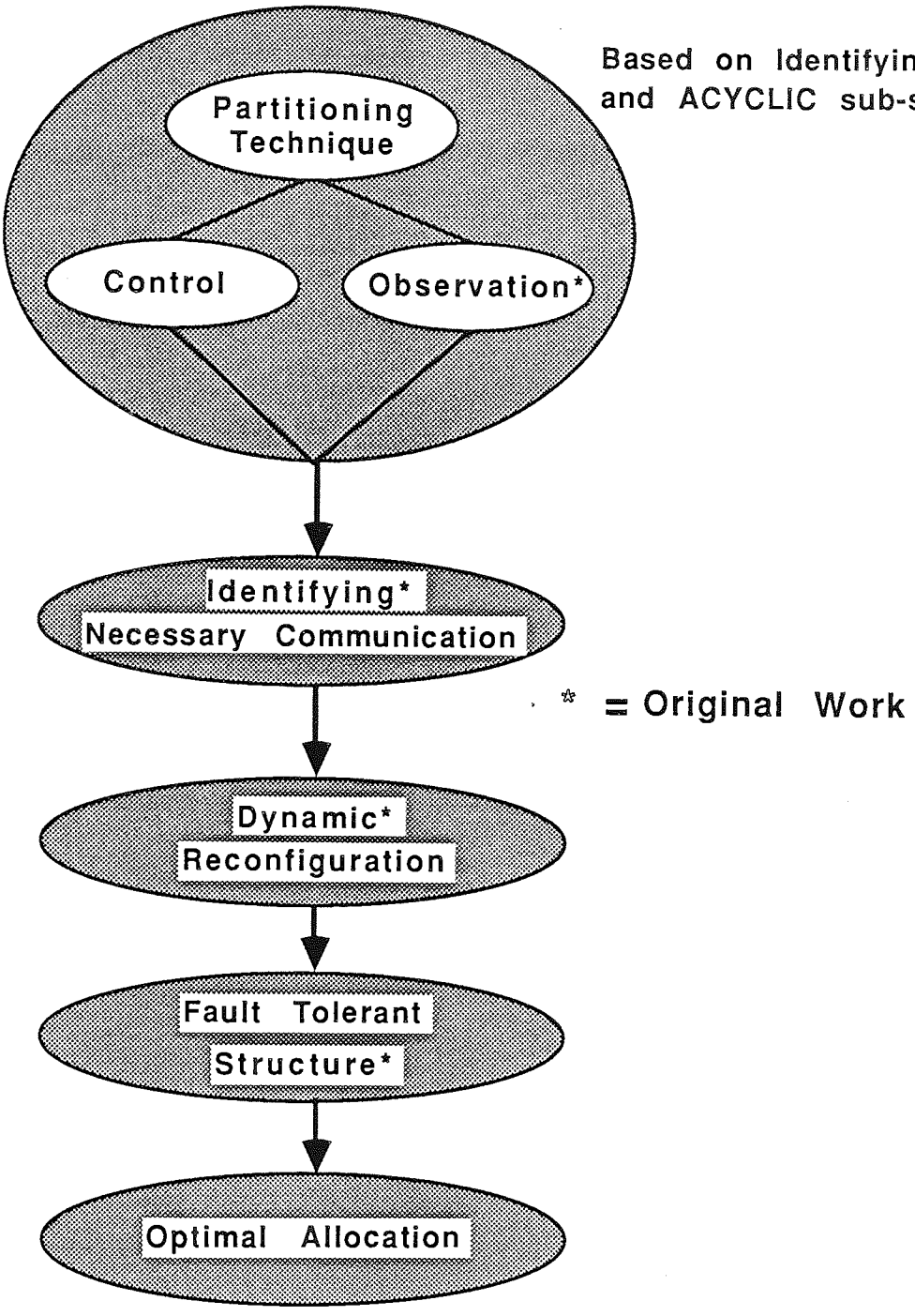


Figure 5.9 Design Methodology

5.5 COMMENT AND CONCLUSION

This chapter describes a method for (i) identification of atomic actions, and (ii) the process of dynamic reconfiguration. It also shows the procedures involved in the design methodology proposed in this thesis.

The following sub-sections will draw conclusions from the work presented in this chapter.

5.5.1 Atomic action

The identification of atomic actions was demonstrated using an example from chapter 4 and the modelling technique of Petri nets. It highlighted the major problem of **protecting a transaction** involved in update/access in a distributed database. A protection mechanism, required for real-time time-critical system transactions, is presented (the **ISO CCR protocol**).

By protecting the processes and transactions in a real-time time-critical system the final integrated product will be made very resilient to failures.

5.5.2 Dynamic reconfiguration

This objective was demonstrated by using the directed graph technique and the potential controllability and observability criteria in the Boolean domain. An important side effect occurred when determining where the processes can be reconfigured dynamically. This analysis identified the system weakness points if the system was to be made resilient against failures of computing elements as well as the external interface to the environment.

. The major problem here is what mechanism is to be used in determining when the dynamic reconfiguration is to take place. Again this requires further research.

5.5.3 Design methodology

The methodology proposed for designing distributed control systems using computers and microprocessors is shown in summarised procedural form in section 5.4. This shows the steps which are necessary to design and develop a decentralised control system which provides graceful, degraded service when failure occurs. The design technique is based on the use of the structural partitioning technique to extract the natural concurrent processes for control software of a linear system.

CHAPTER 6

6.0 CONCLUSION AND FURTHER WORK

6.1 CONCLUSION

The partitioning technique reduces system complexity and therefore is an attractive **tool** for use in analysis and design of large or complex systems. The major objective of the thesis has been the derivation of a design methodology for distributed or decentralised control systems which employs the partitioning technique.

The design methodology is structured around a partitioning technique which makes explicit the natural concurrency of a system. This concurrency is extracted by identifying cyclic and acyclic sub-systems. The state space equations representing a continuous-time linear system are then mapped onto a digraph which is then partitioned into cyclic and acyclic sub-systems.

This decomposition maps directly onto a distributed set of communicating sequential processes which control, observe, and estimate in a noisy environment the state of the partitioned plant. This decomposition also partitions the database associated with the controlled plant at its natural boundaries. The distributed database and communicating sequential processes define the communication structure necessary for distributed control. These sets of logical objects: processes, databases, etc, can be then be optimally allocated to the network of processors with its communication network.

The use of the partitioning technique and correctness proofs will help in the production of error-free software for a large system. Since non-trivial software cannot be exhaustively tested, provision

for hidden design errors must be made. The design must include fault tolerant structures. Atomic structures are identified at the decomposition stage rather than after the coding stage. To achieve a control structure which is resilient, dynamic reconfiguration should be incorporated if at all possible and the fault tolerant design should be incorporated into the final system.

To achieve the aims it was necessary to find a partitioning technique which satisfied all the questions posed in chapter 2: how are the tasks to be partitioned?; what effect do these partitions have on the computation and communication systems?; will the partitions alter the dynamic behaviour of the system?; will they identify and produce a fail safe design? An extensive survey revealed that a variety of techniques existed which satisfied some criteria but not all of them.

The partitioning techniques embedded in **Mascot, JSD, Object Oriented Programming** and others were suitable for particular tasks only. The use of decomposition techniques above depended upon either the user's experience or a mechanical approach. The partitioning technique used by Evans et al [7] used at the structure level to identify cyclic sub-systems for pole placement seemed to be generic to decomposition at a plant level and to the identification of software processes, where resilient structures can be incorporated into the design. This generic technique was used in chapter 4 at the system (plant) level and in chapter 5 at the software/database levels. This partitioning technique used at structure level satisfies all the necessary requirements above and is a generic technique for all levels.

Use of this partitioning technique satisfied four objectives in the design methodology:-

i) The first objective achieved is decentralised observation and estimation of state variables in continuous and discrete-time

systems respectively (chapter 4). Decentralised observation in continuous-time can be achieved by using a simpler numerical technique than the other decentralised techniques [8,9,10]. Also the observation processes are resilient to failures against other observation processes failing, because they are decoupled dynamically.

The approach of decentralised observation was reformulated from the decentralised problem for interconnected systems into a problem of synthesising a decentralised observer assuming that no information transfer was possible. This is termed the **unknown** input observer problem. Only basic matrix criteria were shown. It was shown that techniques such as found in ref [10] could be applied directly to solve those matrices because, of the way the system is partitioned structurally.

The same basic idea was carried over into discrete-time systems to estimate the state values in the presence of input and output noise in a decentralised fashion. This entailed adapting the Kalman Filter technique [11], for the first time, to a number of reduced order inter-communicating Kalman filters as demonstrated in chapter 4. The considerable improvement in speed achieved by this generic partitioning technique on a single computing element can be further improved by using computing elements with true parallel processing capabilities and/or using the systolic array approach [84,85].

ii) The second objective achieved is the systematic identification of the necessary communication for distributed control of observation processes, (section 4.2.2). This augmented the work of Momen and Holding [6] who identified the necessary communication to make the system controllable in a decentralised fashion. To

distinguish this form of communication from the work of Momen et al it was called **Secondary communications.**

iii) The third objective of identifying atomic structures is achieved since the software processes obtained by the partitioning technique are the natural side walls of the conversation used in fault tolerant design, (chapter 5) since the processes constitute atomic actions [12]. The identification of atomic actions is demonstrated by example using the modelling technique of Petri nets.

iv) The fourth objective achieved is the dynamic reconfiguration of processes in certain classes of system, (chapter 5). This objective was demonstrated by using directed graphs and the potential controllability and observability technique. The by-product of determining which processes can be reconfigured dynamically highlights the potential weakness of the hardware design. Therefore, appropriate action can be taken to make the system resilient to failures.

6.2 FURTHER WORK

The partitioning technique was shown to be applicable to linear systems in both the continuous and discrete-time domain. However, it was pointed out in section 4.4 that this partitioning technique is equally applicable to non-linear systems [7]. Although other aspects also need to be investigated for linear systems, before this work can be applied to non-linear systems.

i) The decentralized version of the Kalman filter can be used to estimate the state of certain classes of system. The partitioning technique was applied with additive noise on the input side of the system and on the output sensor from the system. The properties of noise processes were assumed to be white, gaussian, and independent. The next stage of work would be to relax the assumption about the condition of noise processes and to investigate if the system can be still be decomposed and estimated in a decentralized manner. For example, the input and output noise could be correlated. Note this condition results in only minor changes in the operating definition for the **one-shot** Kalman predictors concerned [80].

ii) Only basic matrix criteria for the reliable observer were shown. i.e the unknown input observer problem. Fortunately, due to the partitioning technique the system decomposed will be decoupled dynamically. Therefore, current techniques used in solving the one shot system can be directly applied. The resilience of the sub-observers to failures needs to be demonstrated, first by simulation, and then by implemention on hardware where it can be investigated whether a sub-observer can estimate values for its

state space even if others are injected with faults.

The generic partitioning technique central to the design methodology in the linear system should be investigated for application to **Wiener filtering** [101] and for application to the extended Kalman filter technique used in **self-tuning regulators** [102]. The above ideas and the partitioning technique should be investigated for non-linear system applications i.e. for use in parameter estimation.

When the partitioning technique was used in the decomposition of software, the software processes obtained were found to be atomic structures. The analysis of section 5.2.2 shows that the transactions to and from the distributed database need protection against failures. An interim solution has been provided for the protection of real-time time-critical transactions in a distributed database. Further research is required to identify if the above mechanism is appropriate.

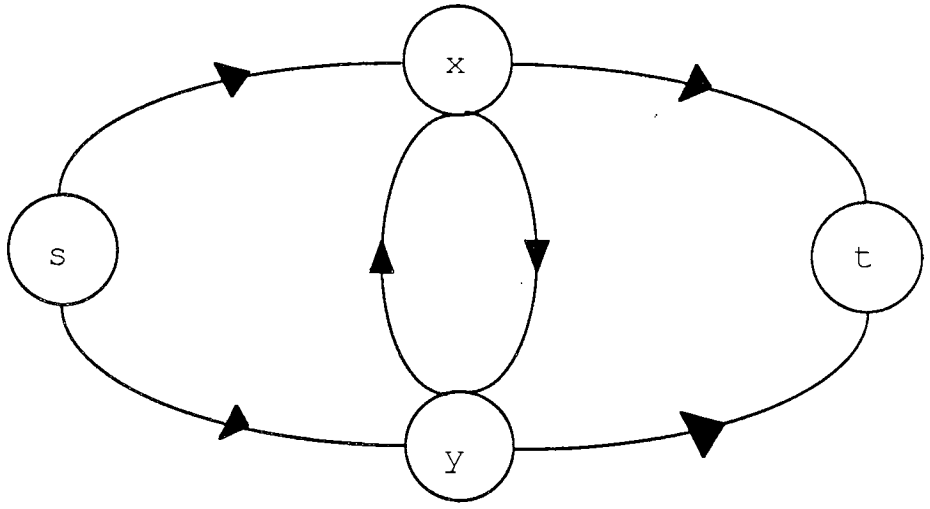
It is felt that the design methodology described in chapter 5 could lend itself to automation. Several stages of design have been implemented using conventional programming languages in the course of this research: decomposition of a system from its state space representation, identifying necessary communication and the optimal allocation process. The aspects of identifying concurrent processes, atomic actions and dynamic reconfiguration can be approached using a Logic programming language because they require artifical intelligence techniques.

APPENDIX A

A.1 Graph Terminology [42,43,52,103]

A directed graph G = [N;A] (or digraph) consists of a collection "N" of elements x, y, ... , (nodes), together with a subset "A" of ordered pairs (x,y) of elements taken from "N" (arcs).

e.g.



N = {s,x,y,t}

A = {(s,x), (s,y), (x,t), (y,t), (x,y), (y,x)}

A **CHAIN** is a sequence of distinct nodes and arcs defined as

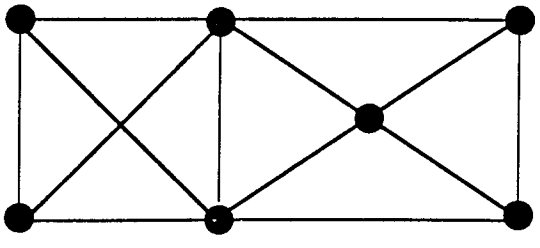
$x_1, (x_1, x_2), x_2, \ldots\ldots, x_{n-1}, (x_{n-1}, x_n), x_n$

e.g. s, (s,x), x, (x,t), t

A **CYCLE** is a chain but $x_1 = x_n$ (Known also as **CYCLIC** graph)

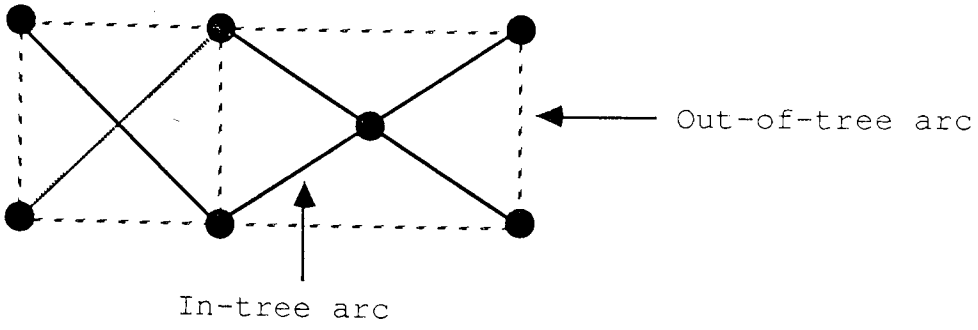
A **TREE** is simply a connected graph that contains no cycle (known also as **ACYCLIC** graph)

A **SPANNING SUBTREE, T** is a tree that spans all the nodes in a connected graph G. It is shown below how to derive a Spanning tree from a given connected graph.



← Connected graph G

By deleting edges we are left with a tree.



we refer to arcs of **T** as **in-tree** arcs, the others as **out-of-tree** arcs.

A **weighted graph** is a graph in which data are associated with the arcs. The value $a(i,j)$ associated with the arc (i,j) is the called the **weight** of (i,j). The $c_{ij}(c(i,j))$ of the directed arc (i,j), called the **capacity** of arc (i,j), is a nonnegitive number. A **flow** in a network assigns a flow in each directed arc which does not exceed the capacity of that arc. Moreover, it is assumed that the flow into a node, v which is neither the source or the sink, is equal to the flow out of v.

MAXIMUM SPANNING (SUB) TREE: a necessary and sufficient condition that a spanning subtree be maximal is that equation A.1 (below) holds for each **out-of-tree** arc

$$a(x_1,x_k) \leq \min[a(x_1,x_2),\ a(x_2,x_3),\ \ldots,\ a(x_{n-1},x_n) \quad ---(A.1)$$

where $a(x_1,\ x_k)$ is the weight assigned to the arc $(x_1,\ x_k)$.

A.2 MAX-FLOW MIN-CUT THEORM

For any network the maximal flow value from source, s to sink, t is equal to the minimal cut capacity of all cuts separating s and t.

The technique is based upon a labelling method, elaborated below. This technique requires a systematic search for an **Augmenting**

Path from s to t. Two points can be noted about the technique:

(i) When the sink is unlabelled, the flow is maximum

(ii) The set of arcs leading from labelled to unlabelled nodes is a minimum cut.

Note: let P (P^-) denote the set of labelled (unlabelled) nodes. (P^- denotes the complement of P). Then the source s is in P and the sink t is in P^-. The set X of arcs (v,w), with $v \in P$ and $w \in P^-$, is called a **cut** and the sum of the capacities of the arcs in X is called the **capacity of the cut**.

$f(x,y)$ and $c(x,y)$ are the actual flow and the maximum capacity of flow on the arc (x,y) respectively.

LABELLING METHOD

Routine A : First the source node receives the label $(-, e(s)=\infty)$. (The source is now **labelled** and unscanned; all other nodes are unlabelled). In general select any labelled, unscanned node x. Suppose it is labelled, $(z\pm, \varepsilon(x))$. To all nodes y that are unlabelled, and such that

$$f(x,y) < c(x,y)$$

assign the label $(x^+, \varepsilon(y))$, where

$\varepsilon(y) = \min [\varepsilon(x), c(x,y) - f(x,y)]$. The label implies that the flow on an arc (x,y) can be increased by an ammount $\varepsilon(y)$.

To all nodes y that are now unlabelled, and such that $f(y,x) > 0$, assign the lable $(x^-, \varepsilon(y))$ where,

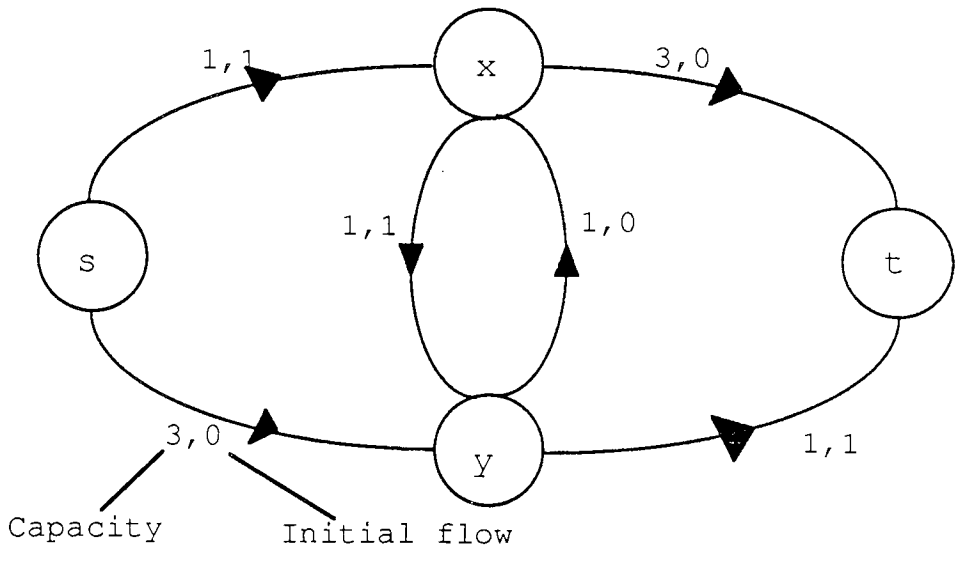
$\varepsilon(y) = \min [\varepsilon(x), f(y,x)]$. The label implies that the flow on an arc (y,x) can be decreased by an ammount $\varepsilon(y)$.

Repeat the general step until either the sink t is labelled and unscanned, or until no more labels can be assigned and sink t is

unlabelled. In the former case, go to **routine B**; in the latter case terminate.

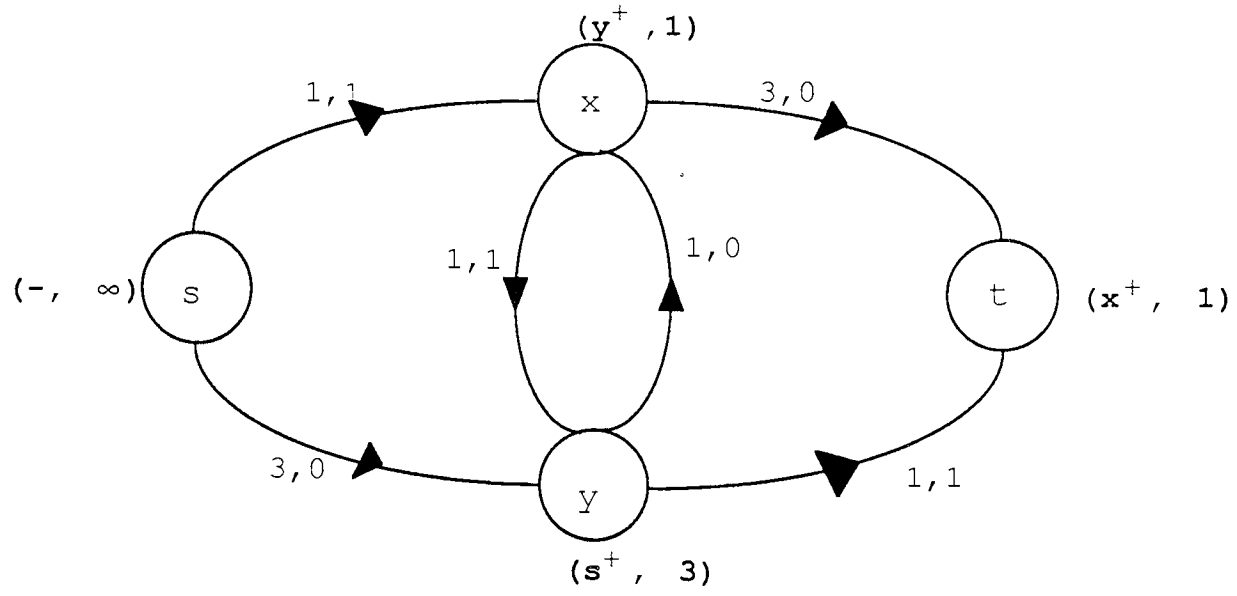
Routine B: The sink t has been labelled $(y\pm, \varepsilon(t))$. If t is labelled $(y^+, \varepsilon(t))$, replace $f(y,t)$ by $f(y,t)+\varepsilon(t)$; if t is labelled $(y^-, \varepsilon(t))$, replace $f(t,y)$ by $f(t,y)-\varepsilon(t)$. In either case, next turn the attention to node y. In general, if y is labelled $(x^+, \varepsilon(y))$, replace $f(x,y)$ by $f(x,y)+\varepsilon(t)$ and if labelled $(x^-, \varepsilon(y))$, replace $f(y,x)$ by $f(y,x)-\varepsilon(t)$, and go onto node x. Stop the flow change when the source is reached, discard the old labels, and go back to **routine A**.

To find a minimal cut for given a digraph with initial flow and maximum capacity flow shown

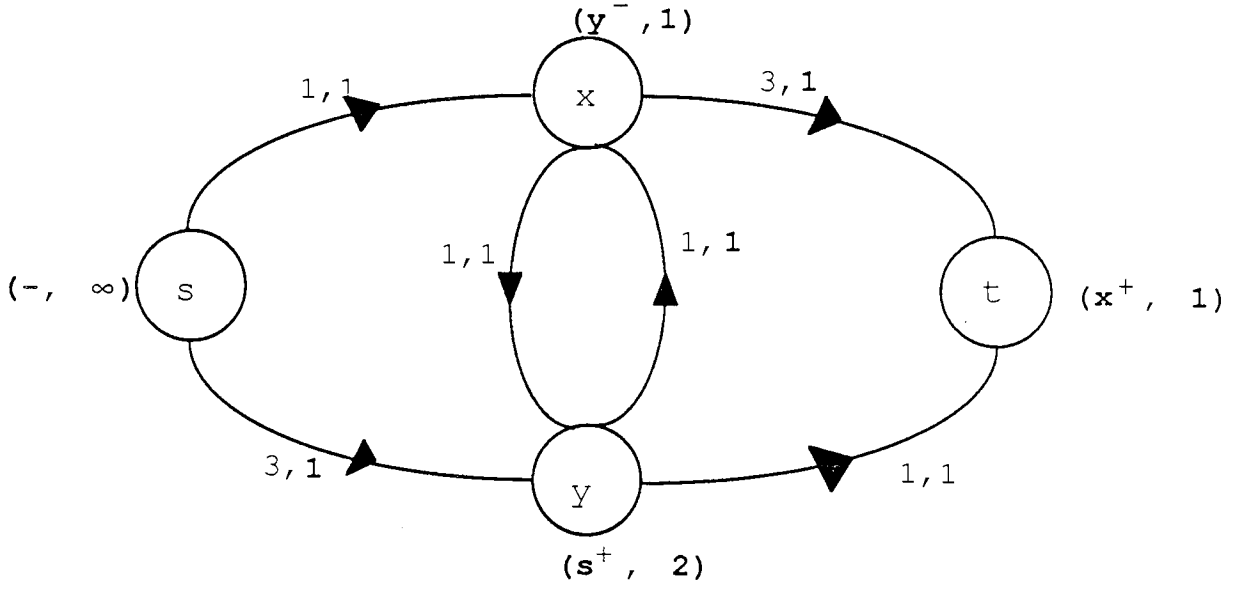


The process starts by initiating **routine A** i.e label source, s $(-, \infty)$ following rules given in the routine we arrive at digraph shown

below.

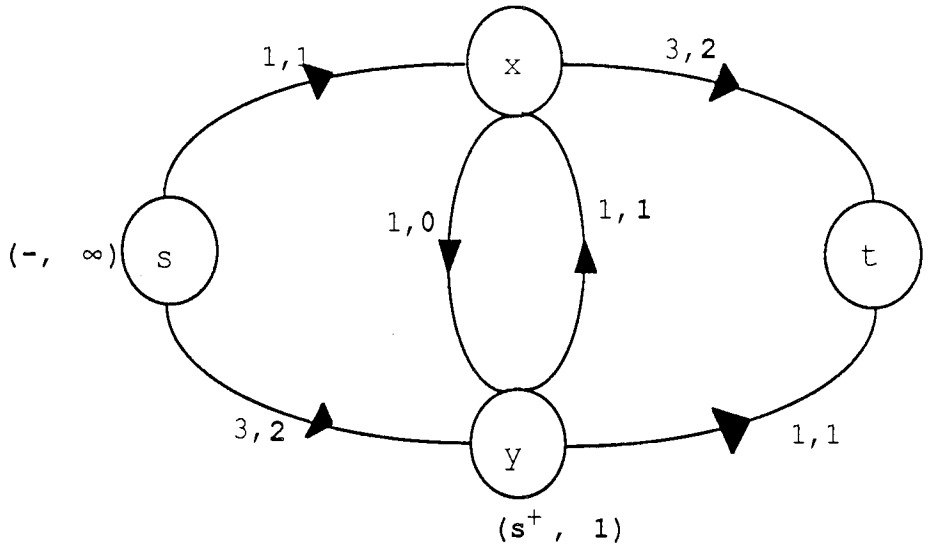


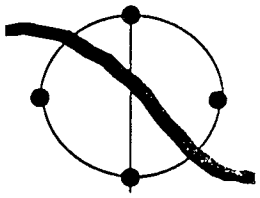
The next step in the process is to initiate **routine B**. The digraph shown below depicts the changes in the flow due to the routine and the re-application of **routine A** again. All the changes to the digraph are higlighted in bold characters.



The final digraph shows that after re-application of **routine B**. The **routine A** is unable to find the flow augmenting path to sink, t.

Therefore the process terminates and we have found the maximum flow minimum cut set.





Minimal cut : (s,x), (y,x) and (y,t)

APPENDIX B

The current work in this thesis is based on design of a control system with resilient control structures; hence, the later work of Momen's [4] was not incorporated into this research. The reason for this omission is that his latter work on necessary communication was based on potential controllable and observable space for each station derived from implicit feedback due to other controllers. The additional steps are detailed here in the same nomenclature as in section 3.2:

Step 6.1: if the intersection of R_{ui} and R_{yj} does not produce a null set then the controllable reachable sub-space of station i becomes:-

$$R^{*}_{ui} = R_{ui} \quad V \quad R_{uj}$$

The reason for this is that part of the controllable reachable sub-space of station i is observed by station j; by implicit feedback, this increases the controllable reachable sub-space of station i. The duality for the observable reachable sub-space also applies.

$$R^{*}_{yj} = R_{yj} \; v \; R_{yi}$$

This process can be further extended as follows even though R_{ui} does not interact with R_{yk}. The controllable reachable sub-space can be increased to include the subspace of station k if and only if

$$R_{ui} \cap R^{t}_{yj} \neq 0 \quad \text{and} \quad R_{uj} \cap R^{t}_{yk} \neq 0$$

then

$$R^{*}_{ui} = R_{ui} \; V \; R_{uj} \; V \; R_{uk}$$

and the duality

$$R^{*}_{yk} = R_{yk} \; V \; R_{yj} \; V \; R_{yi}$$

Step 6.2: The common extended reachable sub-space is found

for all stations

$$R^*_i = R^*_{ui} \cap R^*_{yi}$$

Step 6.3: The extended controllable and observable vector sub-spaces, in collaboration with another station is found for all stations

If $R^*_i \cap R^*_j \neq 0$

then

$$K^*_{ij} = K_i \vee K_j$$

and

$$M^*_{ji} = M_j \vee M_i$$

else

$$K^*_{ij} = K_i \text{ and } M^*_{ji} = M_j$$

Note: the term-rank must be satisfied otherwise similar procedures must be followed as in step 4 of section 3.2. The detailed procedures are given in [4,54].

Step 6.3: The enlarged controllability and observability vector sub-spaces for station i is given by the following equations.

$$K^*_i = \bigcup K^*_{ij}$$

$$M^*_i = \bigcup M^*_{ji}$$

Step 6.4: The common subspace is found for each of the stations

$$L^*_i = K^*_i \cap M^{*t}_i$$

Note for the autonomous version in step 7, section 3.2 is

$$L_i = K_i \cap M^t_i$$

the above steps cater for the implicit feedback approach, the next step for either case is to follow step 8 of section 3.2.

APPENDIX C

C.1 Partitioning based on Flow Relationships

C.1.1 Vertical Partitioning

Vertical partitioning is useful both in centralised and distributed design of a real-time control systems. In a centralised serial processor, vertical partitioning is used to break processing into small tasks. Different threads can be interleaved and no one thread can dominate the Central Processing Unit (CPU). In a distributed system vertical partitioning is also used to segregate tasks appropriate to different processor architectures, to permit geographical distribution, to enhance reconfigurability and reliability options, and to increase performance. The following guidelines have evolved for determining vertical partitions [19]:-

(i) Locate partition boundaries at points where data flows are minimal. These are often correlated with branch points in control flows.

(ii) Branch and rejoin points should be examined since processing in different branches may offer opportunities for use of different hardware structures to enhance performance.

(iii) Recursive loops in processing tend to form natural logical units for partitioning.

C.1.2 Horizontal Partitioning

The major factor which determines whether two sets can be partitioned horizontally is the relationship between their input and

output data spaces, although control flow has some influence. The following guidelines indicate suitable opportunities for horizontal partitioning [19]:-

(i) Completely disjoint processing steps; no data relationship, direct or indirect, between the steps and no mandatory predecessor-successor control flow relationship.

(ii) Instance independence; the data involved in one data processing transaction is independent of the other transactions even though the processing performed is identical.

(iii) Branch and rejoining nodes; in control flow graphs an **AND** branch is an explicit indication of concurrency, and an **OR** branch may indicate an opportunity for horizontal partitioning when the processing on each branch is suitable for a different harware architecture.

C.2 Partitioning Based on Data Access

Rules for data access partitioning and subsequent database organisation can be summarised as follows [19]:-

(i) Separate functions which operate on disjoint sets of data or disjoint regions of a set.

(ii) Separate functions which operate on different spans of data.

(iii) Separate functions which have inherently different time constants and access rates because of the data on which they operate.

(iv) Organise databases onto levels according to the span of the data and degree which nature of higher levels can be hidden from functions which operate on lower levels.

APPENDIX D

Programs written using sequential language can be written using six primitive processes [104]:

INPUT

OUTPUT

ASSIGNMENT

SEQUENCE

SELECTION

REPETITION

Each basic primitive can be modelled as a Petri net, such that the state of the process is represented by the marking of the places of the net, μ. A formal definition of a Petri net can be found in [91].

The Petri net models for these primitive constructs are shown in **figure** D.1. It follows that any sequential program can be modelled as a Petri net by combining a number of these primitives.

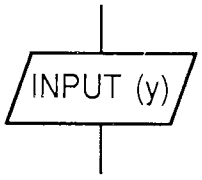
From the above it can be seen that every place has a unique output transition, except for places which preceed decisions *(selection, repetition)*; these places have two output transitions corresponding to TRUE and FALSE outcome of the decision predicate. The choice as to which arc to take can be made non-deterministically or by some outside influence.

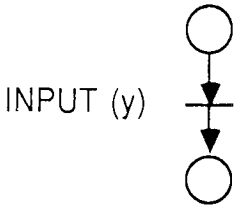
The sequential constructs described above are sufficient to describe sequential systems and the sequential parts of concurrent systems. However, concurrent systems can not be fully described using the sequential concepts alone. Additional constructs must be introduced to describe parallelism, inter-process communications and inter-process synchronisation [105].

Process	Program	Flow Diagram	Petri Net Model

INPUT

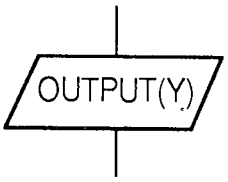
INPUT (y)

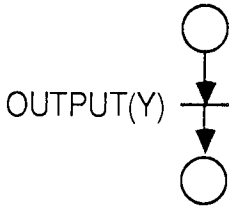




OUTPUT

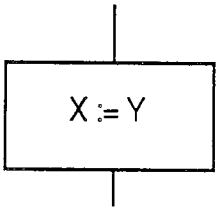
OUTPUT (Y)

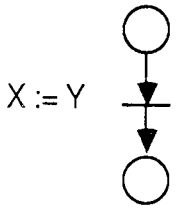




ASSIGNMENT

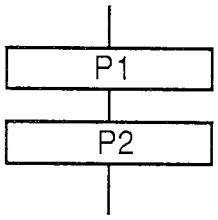
X := Y

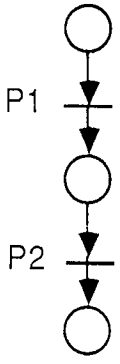




SEQUENCE

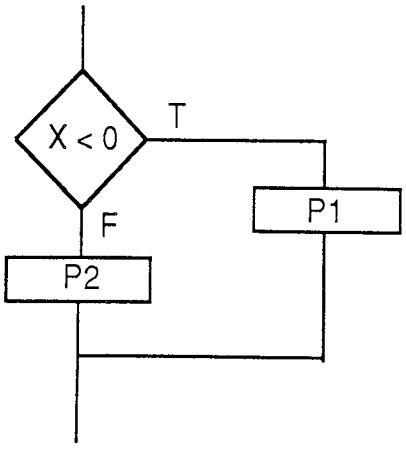
SEQ
P1
P2

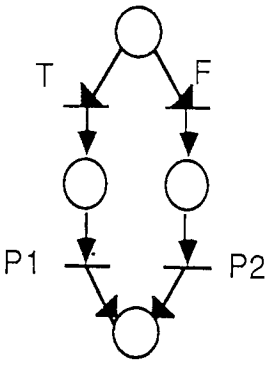




SELECTION

IF X < 0 THEN P1
ELSE P2





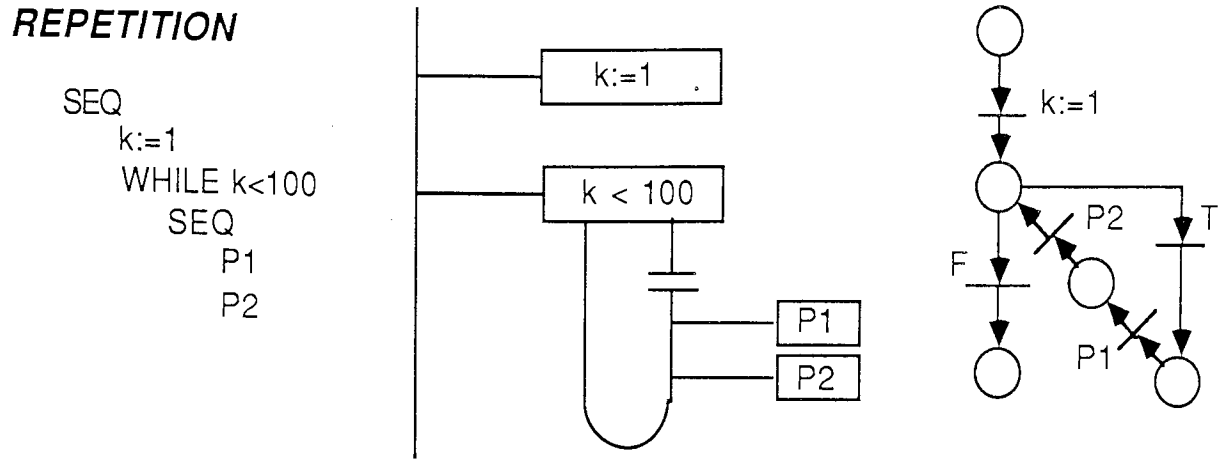
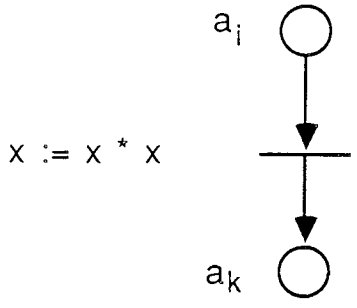


Fig. D1 Petri Net Models of Sequential Software Constructs.

Concurrent programming languages such as Occam allow the user to model concurrent systems. Such systems are composed of separate, interacting components. Each component may itself be a process, and its behaviour can be independent of the other components of the system, except for well-defined interactions with other components. To deal with concurrent systems Petri net models were developed [16] for concurrent constructs such as parallel processes, synchronised communications and asynchronous **ALT** processses.

Assignment

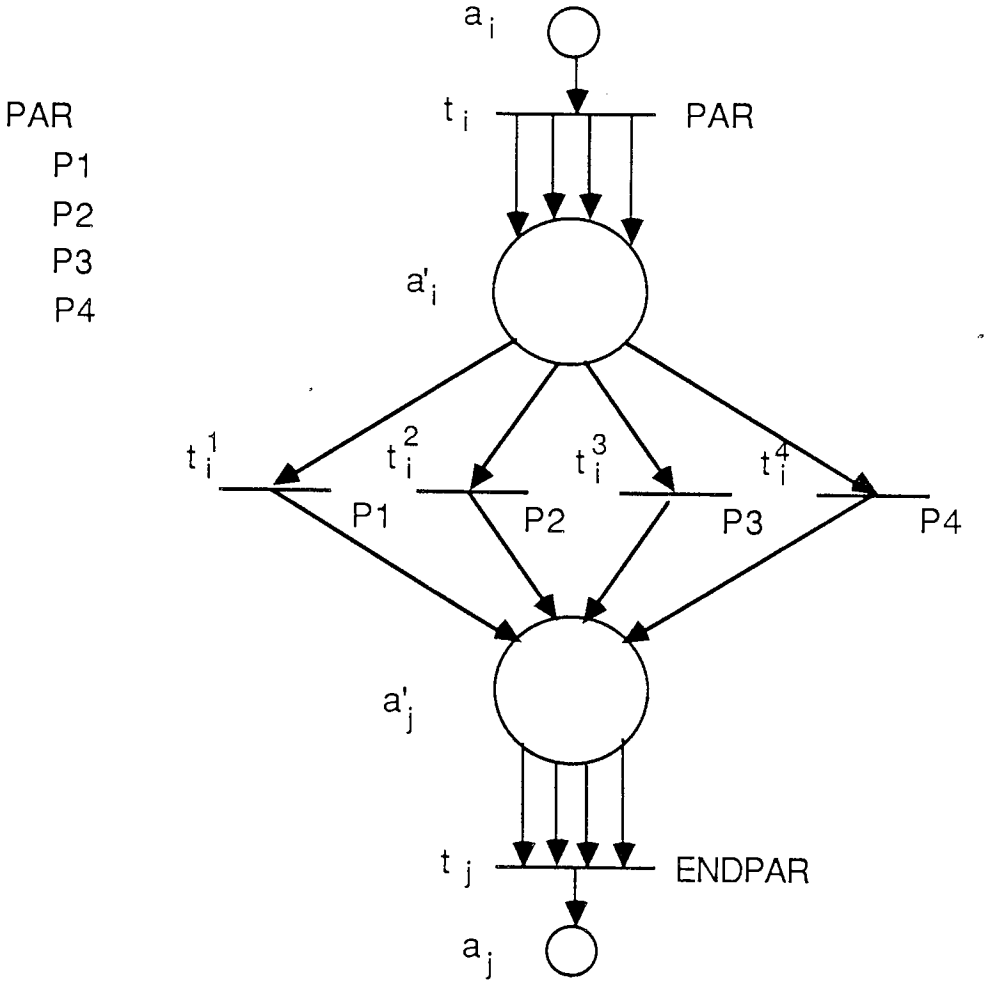
Assignment is an action which involves a single process only: it can be modelled as a transition with single input and output arcs.



Parallel

In the parallel construct all actions are initiated

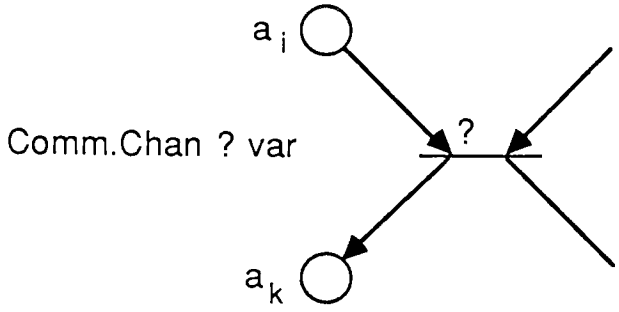
simultaneously. The construct does not terminate until all parallel processes have terminated.



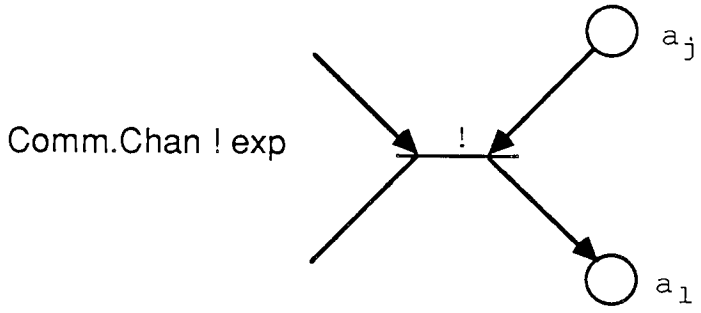
Communications

For the action of communication two processes are involved. One sends the information and one recieves it. Thus a transition modelling a communication requires at least two output arcs, again one to each process. To distinguish input and output actions the occam notation for input and output on the transitions is used.

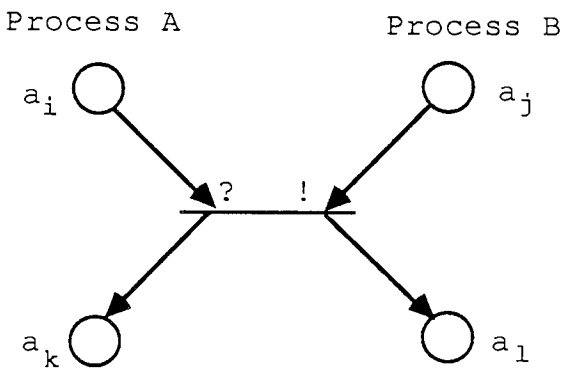
Input



output

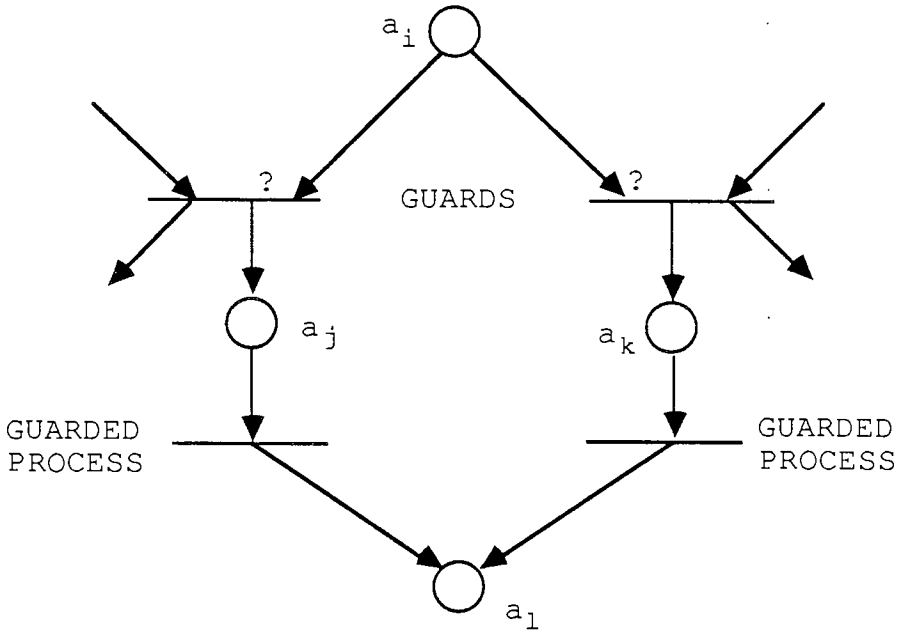


These two actions, input and output, always appear in pairs in the system. In systems which use a parallel processing language such as Occam where communications are synchronised, the same transition will be shared by both processes involved in the communication.



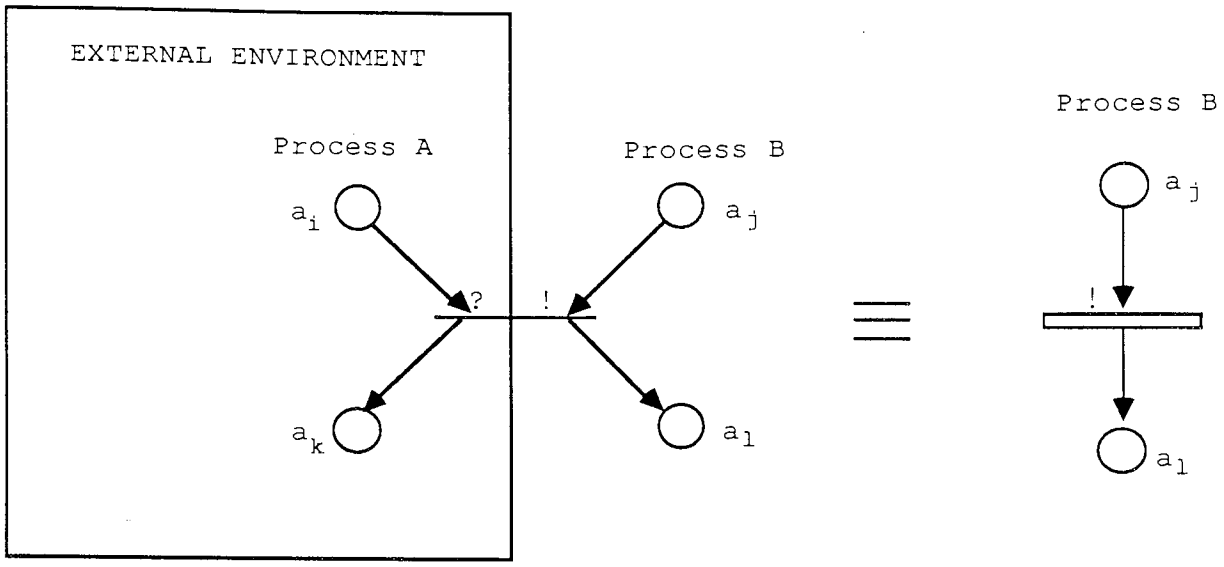
ALT (alternative construct)

The alternative construct chooses one of its components for execution. Each component process has a guard which is an input (?). The process whose guard is satisfied earliest is executed. If more than one guard is satisfied the choice as to which alternative is taken is defined as being arbitrary. This construct can be modeled by the Petri net shown below



Interface to Environment

In order to keep the transformation from Occam program to Petri net model as simple as possible, the communication to an external environment (i.e. output to the environment) is simplified as shown in the diagram below:


EXTERNAL ENVIRONMENT
Process A
Process B
a_i
a_j
?
!
a_k
a_l
Process B
a_j
!
a_l

REFERENCES

[1] Javdan, M.R. and Richards, R.J., (1977). Decentralized control system theory - A critical evaluation. INT. J. CONTROL, Vol. 26, No. 1 pp 129-144.

[2] Kron, G., (1953). A Set of Principles to Interconnect the Solutions of Physical System. J. Appl. Physics, Vol. 24, No. 8, August, 1953 pp 965-980.

[3] Dantzig, G. and Wolfe, P., (1960). Decomposition Principles for Linear Programs. Op. Res., Vol. 8, pp 101-111.

[4] Momen, S.E.M., (1982). Structural Controllability and Stabilization of Decentralised system. Ph.D Thesis, Dept. E. and E. Eng., Queen Mary College, Univ. of Lon.

[5] Harry, F., (1959). A Graph-Theoretic Method for the Complete Reduction of a Matrix with a View Toward Finding its Eigenvalues. J. Maths and Physics Vol. 38, pp 104-111.

[6] Momen, S.E.M. and Holding, D.J., (1981). Control and Communication structures in distributed control systems. IEE International Conference on Control and its Applications. Warwick, UK., Pub. No. 194, pp 291-296.

[7] Evans, F.J. and Schizas, C., (1981). Control System Design using Graphical technique. IEE PROC. Vol. 128, Pt. D, No. 3, pp. 77-84.

[8] Siljak, D.D. and Vukcevic, M.B., (1978). On decentralized estimation. Int. J. Control Vol. 27, No. 1, Jan 1978, pp. 113-132.

[9] Geromel, J.C. and Yamakami, A., (1982). Stabilization of continuous and discrete linear systems subjected to control structure constraints. Int. J. Control Vol. 63, No. 3, pp 429-444.

[10] Viswanadham, N. and Ramakrishna, A., (1982). Decentralized estimation and control for interconnected systems. Large Scale Systems Vol. 3, pp. 255-266.

[11] Kalman, R.E. and Bucy, R., (1961). New results in Linear Filtering and Prediction Theory. J. of Basic Eng., Trans. ASME, series D, Vol. 83, No. 3, pp 95-108, 1961.

[12] Anderson, T., and Lee, P.A., (1981). Fault Tolerance Principles and Practice. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[13] Hecht, H. (1979). Fault Tolerant Software. IEEE Trans. on Reliability, Vol. R - 28, No. 3, Aug. 1979, pp 227-232.

[14] Lomet, D.B., (1977). Process Structuring, Synchronization and Recovery using Atomic Actions. Sigplan Notices, Vol. 12, No. 3, March 1977, pp 128-137.

[15] Randell, B., (1975). System Structure for Software Fault Tolerance. IEEE Trans. on Soft. Eng., Vol. SE-1, No. 2, June '75, pp 220-232.

[16] Tyrrell, A.M., (1987). The Design of Fault Tolerant Software for Loosely Coupled Distributed Systems. Ph.D. Thesis, Dept. of Elect. & Electronic Eng. and Applied Physics, Aston University.

[17] Occam Programming Manual (1984). Prentice-Hall.

[18] Holding, D.J., Carpenter, G.F. and Tyrrell, A.M., (1984). Aspects of Software Engineering for Systems with Safety Implications. Eurocon, Brighton '84, pp 235-239.

[19] Lawson, J.T. and Mariani, M.P., (1978). Distributed data processsing system design - A look at the paritioning problem. INVITED PAPERS COMPSAC 78, Chicago, Illinois, pp 358-363.

[20] Bell, D., Morrey, I. and Pugh J., (1987). SOFTWARE ENGINEERING A Programming Approach Prentice/Hall International

[21] Sommerville, I., (1985). Software Engineering 2nd edition, Addison-Wesley.

[22] The official Handbook of MASCOT, (1983). MASCOT II, Issue 2, March 1983. Joint IECCA and MUF Commitee on MASCOT 1983.

[23] Dowson, M., (1986). ISTAR - an Integrated Project support Environment. Proceedings of 2nd ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, Palo Alto CA, Dec. 1986.

[24] Jones, C.B., (1980). Software Development - the Rigorous Method. Prentice-Hall, 1980.

[25] Hoare, C.A.R., (1985). Communicating Sequential Processes. The Prentice-hall International Series in Computer Science.

[26] Wirth, N., (1971). Program development by stepwise refinement. Communications of the ACM, Vol. 14, No. 4, 1971.

[27] Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R., (1972). Structured Programming. Academic Press, 1972.

[28] Yourdon, E., Myers, G. and Constantine L.L., (1979). Structured Design Prentice-Hall, Englewood Cliffs, N.J, 1979.

[29] Pressman, R.S., (1987). SOFTWARE ENGINEERING - A Practitioner's Approach. McGraw-Hill International Editions in Computer Science Series, 1987.

[30] Jackson, M., (1975). Principles of Program Design. Academic Press, 1975.

[31] Software Tools for Application to Large Real Time Systems, Department of Trade and Industry, 1984.

[32] Jackson, M., (1983). System Development. Prentice-Hall International, 1983.

[33] Simpson, H.R. and Jackson K., (1979). Process synchronisation in MASCOT. The Computer Journal, Vol. 13, No. 2, pp 115-134, 1979.

[34] Robinson, D., (1981). Object-Oriented Software Systems. Byte publication, Vol. 6, No. 8, pp 76-86, Aug. 1981.

[35] Robinson, D. and Goldberg A., (1981). The smalltalk-80 System. Byte publication, Vol. 6, No. 8, pp 36-48, Aug. 1981.

[36] Jensen, E.D. and Boebert, W.E., (1976). Partitioning and Assignment of distributed processing software. COMPCON 76 EAST, pp 348-352.

[37] Ramamoorthy, C.V. and Krishnaro, T., (1976). Design Issues in Distributed Computer Systems. Distributed System - Infotech State of the Art Report.

[38] Parnas, L.D., (1972). On the Criteria to be Used in Decomposing system into modules. Comm. of ACM, Vol.15, No. 12, pp 1053-1058.

[39] Hamilton, M. and Zeldin, S., (1976). Higher order Software - A Methodology for Defining Software. IEEE S.E. 2, No.1, March 1976, pp 9-32.

[40] Jenny, C.J. and Haessig, K., (1980). Partitioning and Allocating Computational objects in Distributed Computing System. Proc. IFIP Congress 80, Melbourne, Australia, pp 593-598.

[41] Stone, H.S. and Bokhari, S.H., (1978). Control of Distributed Processes. Computer, Vol. 11 No. 7, pp 97-106.

[42] Reingold, E.M., Nievergelt, J. and Deo, N., (1972). Combinatorial algorithms: theory and Practice. Prentice-Hall, Inc.

[43] Ford, L.R. and Fulkerson, D.R., (1962). Flows in Network. Princeton University Press, princeton, N.J.

[44] Jenny, C.J., (1982). On the Placement of files and processes in a system with decentralized intelligence. Int. Zurich Seminar on digital communications man-machine interaction, Zurich, Switzerland, pp B1.1 - B1.8.

[45] Bokhari, S.H., (1979). Dual processor scheduling with dynamic reassignment. IEEE Trans. on Soft. Eng., Vol. SE-5, NO. 4, pp 341-349.

[46] Gonzalez, M.J. and Ramamoorthy, C.V., (1969). A survey of techniques for recognising parallel processable streams in computer programs. In fall Joint Comput. Conf., AFIPS Conf. Proc., Vol. 35, pp 1-17.

[47] Gonzalez, M.J. and Ramamoorthy, C.V., (1972). Parallel Task Execution in a Decentralised System. IEEE Trans. on Comput., Vol. C-21, pp 1310-1322.

[48] Ramamoorthy, C.V., (1966). Analysis of graphs by connectivity considerations. Journal of ACM, Vol. 13, No. 2, April 1966, pp 211-222.

[49] Weist, J.D. and Levy, F.K., (1969). Management guide to PERT/CMP. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[50] Kevorkian, A.K., (1975). Structural aspect of large scale systems. Paper 193, presented at 6th IFAC world congress, Boston, MA, USA.

[51] Franksen, O.I., Falster, P. and Evans, F.J., (1979). Qualitative Aspect of Large Scale System. Lecture notes in Control and Information Science, 17, Springer-verlag.

[52] Roberts, F.S., (1976). DISCRETE MATHEMATICAL MODELS - with application to social, biological, and environmental problems. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[53] Luce, D.E.,(1952). A note on Boolean matrix Theory. Proc. AM. Math. Soc., Vol. 3, pp 382-388.

[54] Momen, S.E.M. and Evans, F.J., (1983). Structurally fixed modes in decentralised system, part I and II. IEE PROC., Vol. 130, pt. D, No. 6, pp 313-327.

[55] Sezer, M.E. and Siljak, D.D., (1981). Structural fixed modes. Syst. & Control lett., 1, pp 60-64.

[56] Schields, R.W. and Pearson, J.B., (1976). Structural controllability of multi-input linear systems. IEEE Trans., AC-21, pp. 203-212.

[57] Schizaz, C. and Evans, F.J. (1981). A graph theoretic approach to multi-variable control system design. Automata, 17, pp 371-377.

[58] Gilman, L and Rose, A.J., (1984). APL: an interactive approach. John Wiley and Sons, Inc.

[59] Kobayashi, H., Hauafusa, T. and Yoshikawa, T., (1978). Controllability under Decentralised Control, Vol AC-23, pp 182-188.

[60] Wonham, W.M., (1974). Linear multivariable control, a multivariable approach. Lecture notes in economics and mathematical Systems, Vol 101, Springer-verlag.

[61] Tyrrell, A.M. and Holding, D.J., (1986). Design of Reliable Software in Distributed Systems Using the Conversation Scheme. IEEE Trans. on Soft. Eng., Vol. SE-12, No. 9, Sept., '86, pp 921-928.

[62] Anderson, T., Lee, P.A. and Shrivastava, S.K., (1978). A Model of Recoverability in Multilevel Systems. IEEE Trans. on Soft. Eng., Vol. SE - 4, No. 6, Nov. 1978.

[63] Pierce, W.H., (1965). Fault - Tolerant Computer Design. New York, Academic Press, 1965.

[64] Randell, B., Lee, P.A. and Treleaven, P.C., (1978). Reliability Issues in Computing System Design. Computing Surveys, Vol. 10, No. 2, June 1978, pp 123-165.

[65] Campbell, R.H., Anderson, T. and Randell, B. (1983). Practical Fault Tolerant Software for Asynchronous Systems. Proc. Conf. IFAC Safecomp '83, Cambridge 1983, pp 59-65.

[66] Horning, J.J., Lauer, H.C., Melliar-Smith, P.M., and Randell, B., (1974). A Program Structure for Error Detection and Recovery. Proc Conf.Operating Systems: Theoretical and Practicle Aspects, April 1974, pp 177-193.

[67] Chen, L. and Avizenis, A., (1978). N-version Programming: A Fault Tolerance Approach to reliability of Software Operation. Digest of 8th Annual International Conference on Fault Tolerant Computing, Toulouse, June 1978, pp 3-9.

[68] Russel, D.L. and Tiedman, M.J., (1979). Multiprocess Recovery using conversations. Proc. FTC - 9, pp 106-109.

[69] Kim, K.H., (1982). Approaches to Mechanization of the Conversation Scheme Based on Monitors. IEEE Trans. on Soft. Eng., Vol. SE - 8, No.3, May 1982, pp 189-197.

[70] Boebert, W.E., Franta, W.R., Jensen, E.D. and Kain, R.Y., (1978). Decentralised Executive Control in Distributed Computer Systems. Proc. of COMPAC 78 Computer Software and Application Conference, Chicago, I.L., U.S.A. 13-16th Nov. 1978, pp 254-258.

[71] Grasso, P.A., Forward, K.E. and Dilion, T.S., (1982). Operating system for a dedicated common memory multimicroprocessor system. IEE Proc. E., Comput. and Digital Tech., Vol. 129, No. 5, pp 200-206.

[72] Mehra, S.K. and Prof. Majithai, J.C., (1982). Reconfigurable computer architectures. IEE Proc. E., Comput. and Digital Tech., Vol. 129, No. 4, pp 156-164.

[73] Kramer J. and Magee, J., (1985). Dynamic configuration for distributed systems. IEEE Trans. on Software Eng., Vol. SE-11, No. 4, pp 424-435.

[74] Nelson, B.J., (1981). Remote Procedure Call. Xerox Palo Alto reseaach Centre, CSL-81-9, May 1981.

[75] LeBlanc, R.J. and Maccabe, A.B., (1982). The Design of a Programming Language Based on Connectivity Networks. Proceedings of the Third International Conference on Distributed Computing Systems, Tampa, Florida, 1982.

[76] Halsall, F., Grimsdale, R.L., Shoja, G.C. and Lambert, J.E., (1983). Development environment for the design and test of application software for a distributed multiprocessor computer system. IEE Proc. E., Comput. and Digital Tech., Vol. 130, No. 1, pp 25-31.

[77] Kramer, J., Magee, J., Solman, M. and Lister, A., (1983). CONIC: an integrated approach to distributed control systems. IEE Proc. E., Comput. and Digital Tech., 1983, Vol. 130, No. 1, pp 1-10.

[78] Sandell, Jr. N.R., Varaiya, P., Athans, M. and Safnov, M.G., (1978). Survey of decentralised control methods for large-scale systems. IEEE Trans. Automatic Control Vol. 23, No. 2, pp. 108-128.

[79] Luenberger, D.G., (1971). An Introduction to Observers. IEEE Trans. on Automatic Control, Vol. AC-16, No. 6, Dec. 1971, pp. 596-602.

[80] Warwick, K., (1986). Observers, State Estimation and Prediction. Lecture Notes in Control and Information Science, 79, Signal Processing for Control, Springer-Verlag, pp 245-261.

[81] Anderson, B.D.O. and Moore, J.B., (1979). Optimal Filtering. Prentice-Hall Inc., Englewood Cliffs, 1979.

[82] Khun, U. and Schmidt, G., (1984). Decentralized observation: A unifying presentation of five basic schemes. Large Scale Systems Vol. 7, pp. 17-31.

[83] Evans, F.J. and Schizas, C., (1981). APL and Graph Theory in dynamic system analysis. IEE PROC. Vol. 128, Pt. D, No. 3, pp. 85-92.

[84] Irwin, G.W. and Rogers, E., (1987). Novel algorithm and architectures for Kalman Filtering. IEE colloqium on "Parallel Processing : A new direction for control?". Savoy Place, London, 6th Febuary, 1987.

[85] Irwin, G.W. and Gaston F.M.F., (1987). Occam simulation of a Systolic Architecture for Parallel Kalman filtering. IEE Workshop on Parallel processing in control-the transputer and other architectures. Bangor, 20-22 September, 1987.

[86] Kailath, T., (1980). Linear Multivariable Systems. Springer-Verlag, New York, 1974.

[87] M^{C}lean, D., (1978). Gust-Alleviation Control System for Aircraft. IEE PROC. Vol. 125, No. 7, July 1978, pp 675-685.

[88] Orr, C.H., (1980). System aggregation technique for control system design. Ph.D Thesis, Dept. E and E. Eng., Queen Mary College, Univ. of Lon., UK.

[89] Orr, C.H. (1976). The design of controllers for a power station boiler using the dyadic control method. M.Sc dissertation, Dept. E. and E. Eng., Queen Mary College, Univ. of Lon., UK.

[90] Welsh, J. and McKeag, M., (1980). Structured System Programming. Prentice-Hall.

[91] Peterson, J.L., (1980). Petri Net Theory and the Modelling of Systems. Prentice-Hall.

[92] Yau, S.S. and Shatz, S.M., (1982). On Communication in the Design of Software Components of Distributed Computer Systems. 3rd Int. Conf. on Distributed Computing Systems, 1982, pp 280-287.

[93] Hoare, C.A.R., (1978). Communicating Sequential Processes. Comm. ACM, Vol. 21, No. 8, 1978, pp 666-677.

[94] Heimerdinger, W.L., (1978). A Petri Net Approach to System Level Fault Tolerance Analysis. Proc. of the National Electronic Conference, Vol. 32, 1978, pp 161-165.

[95] Han, Y.W., (1978). Performance Evalution of a Digital System usng Petri Net - Like Approach. Proc. of the National Electronics Confrence, Vol. 32, 1978, pp 166-172.

[96] Sloman, M, and Andripoulos, X., (1983). Databases for Real-Time Application. Research Report DOC 83/9 in Dept. of Computing, Imperial College, London SW7 2BZ, U.K., March 1983.

[97] Delobel, C. and Casey, R.G., (1973). Decomposition of a Data Base and the Theory of Boolean Switching Function. IBM J. Res. Develop., Vol. 17, No. 5 pp 374-386.

[98] Spector, A.Z. and Schwarz, P.M., (1983). Transactions: A Construct for Reliable Distributed Computing. Operating Systems Review (USA), Vol. 17, No. 2, April 1983, pp 18-35.

[99] International Organisation for Standardisation (ISO): DIS 8650/3 - Information Processing Systems - Open System Interconnection - Definition of Common Application Service Elements - Part 3: Commitment, Concurrency and Recovery: Protocol Specification.

[100] Carpenter, G.F., Tyrrell, A.M. and Holding, D.J., (1986). Guidelines for the Synthesis of Software for Distributed Processes. PES3 Conference Guernsey 1986 pp. 164-175.

[101] Bozic, S.M., (1979). Digital and Kalman filtering. Edward Arnold (Publishers) Limited.

[102] Warwick, K., (1981). Self-tuning regulators - a state space approach. Int. J. Control, Vol. 33, No. 5, pp. 839-858, 1981.

[103] Johnsonbaugh, R., (1986). Discrete Mathematics. Macmillan Publishing Company, NY, 1986.

[104] Dijkstra, E.W., (1972). Notes on Structured Programming. Academic Press, 1972.

[105] Hansen, P.B., (1977). The Architecture of Concurrent Programs. Prentice-Hall, Englewood Cliffs, NJ, 1977.