Introduction

Present work is the result of one particular idea around a condition established as unknown scenario named fault scenario over system performance. In that respect control law is modified in order to be capable to overcome this unknown scenario. In order to perform this task several issues need to be considered such as fault diagnosis and time delays boundaries.

Several strategies have been reviewed in order to come up with the implementation and integration of three main stages named as fault diagnosis and related heuristic confidence value, fault tolerance strategy and the related real-time scheduling approach and those control strategies suitable for different scenarios considering fault presence and time delays.

The objective of this work is to present to the reader a way to perform reconfigurable control on-line without jeopardize the safety and the stability of the system. This book is focused for undergraduate and postgraduate students interested in reconfigurable control as a strategy to overcome local fault conditions and performance degradation during still manageable fault situations.

This book is divided in five chapters, first chapter is a basic review of communication networks, second chapter presents how real-time systems can define time delays, third chapter gives a review of fault diagnosis and how to use this techniques for health treatment, fourth chapter presents control reconfiguration strategies based upon a review of network control systems, final chapter gives some implementing examples.

CHAPTER I

NETWORK BACKGROUND

1.1.Background

During this chapter a general review of several databuses that conform the most common strategies in terms of inter-process communication is given. This review allows the understanding of common databus behaviour presenting how time delays are the results of data transfer. The objective of this chapter is to show how different protocols perform communication in order to understand communication time delays which are reviewed in following chapters.

1.2.Review of OSI Layer

One of the key issues on distributed systems is the protocol where the integration of the information to be transmitted through the network is conformed. There are several points to be developed into that respect. For instance, the number of OSI layers to be involved with a direct repercussion on user applications.

A distributed system is one in which several autonomous processors and data stores supporting processes and/or databases interact in order to cooperate and achieve an overall goal. The processes co-ordinate their activities and exchange information by means of information transferred over a communication network (Sloman et al., 1987). One of the basic characteristics of distributed systems is that interprocess messages are subject to variable delays and failure. There is a defined time between occurrence of an event and its availability for observation at some other point.

The simplest view of the structure of a distributed system is that it consists of a set of physical distributed computer stations interconnected by some communications network. Each station has the capability for processing and storing data, and may have connections to external devices. Table 1.1 is a summary to provide an impression of the functions performed by each layer in a typical distributed system (Sloman et al., 1987). It is important to highlight that this is just a first attempt in order to define an overall formal concept of the OSI layer.

Layer	Example
Application software	Monitoring and control modules
Utilities	File transfer, device handlers
Local management	Software process management
Kernel	Multitasking, I/O drivers, memory management
Hardware	Processors, memory I/O devices
Communication system	Virtual circuits, network routing, flow control error control

Table 1.1 OSI Layer non-formal attempt

This local layered structure is the first attempt in understanding how a distributed system is constructed. It provides a basis for describing the functions performed and services offered at a station. The basic idea of layering is that, regardless of station boundaries, each layer adds value to the services provided by the set of lower layers. Viewed from above, a particular layer and the ones below it may be considered to be a 'black box', which implements a set of functions in order to provide a service. A protocol is the set of rules governing communication between the entities, which constitute a particular layer. An interface between two layers defines the means by which one local layer makes use of services provided by the lower layer. It defines the rules and formats for exchanging information across the boundary between adjacent layers within a single station.

The communication system at a station is responsible for transporting system and application messages to/from that station. It accepts messages from the station software, and prepares them for transmission via a shared network interface. It also receives messages from the network and prepares them for receipt by the station software.

In 1977 the International Standard Organisation (ISO) started working on a reference model for open system interconnection. The ISO model defines the seven layers as shown in Fig 1.1. The emphasis of the ISO work is to allow interconnection of independent mainframes rather than distributed processing. The current version of the model only considers point-to-point connections between two equal entities.

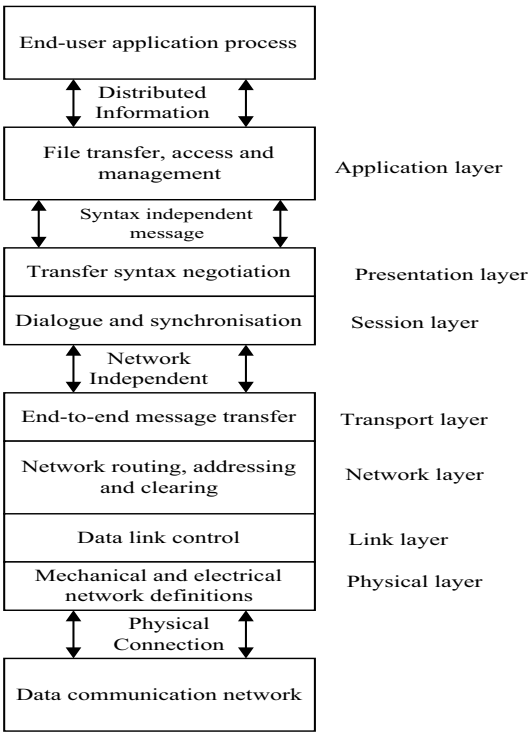

Fig. 1.1 OSI Layers

Application Layer

Those application entities performing local activities are not considered part of the model. A distributed system would not make this distinction as any entity can potentially communicate with local or remote similar entities. The application layer includes all entities, which represent human users or devices, or perform an application function.

Presentation layer

The purpose of the presentation layer is to resolve differences in information representation between application entities. It allows communication between application entities running on different computers or implemented using programming languages. This layer is concerned with data transformation, formatting, structuring, encryption and compression. Many of these functions are application dependent and are often performed by high-level language compilers, so the borderline between presentation and application layers is not clear.

Session layer

This layer provides the facilities to support and maintain sessions between application entities. Sessions may extend over a long time interval involving many message interactions or be very short involving one or two messages.

Transport layer

The transport layer is the boundary between what are considered the application-oriented layers and the communication-oriented layer. This is the lowest layer using an end-station-to-end-station protocol. It isolates higher layers from concerns such as how reliable and cost-effective transfer of data is actually achieved. The transport layers usually provide multiplexing; end-to-end error and flow control, fragmenting and reassembly of large

messages into network packets and mapping of transport-layer identifiers onto network addresses.

Network layer
The network layer isolates the higher layers from routing and switching considerations. The network layer masks the transport layer from all the peculiarities of the actual transfer medium: whether a point-to-point link, packet switched network, LAN or even interconnected networks. It is the network layer's responsibility to get a message from a source station to the destination station across an arbitrary network topology.

Data-link layer
The task of this layer is to take the raw physical circuit and convert it into a point-to-point link that appears relatively error free to the network layer. It usually entails error and flow control but many local area networks have low intrinsic error rates and so do not include error correction.

Physical layer
This layer is concerned with transmission of bits over a physical circuit. It performs all functions associated with signalling, modulation and bit synchronisation. It may perform error detection by signal quality monitoring.

There are several types, which are based on their applications, type of physical networks and specific demands such fault tolerance or communication performance. The classification pursued in this work is related to the used of communication networks. There are two main divisions into that respect general purposes networks and industrial networks. These are characterized for the environment to be proposed. A general description of these protocols is listed next (Lönn, 1999):

- Collision Sense Multiple Access/ Carrier Detect CSMA/CD – IEEE 802.3 Ethernet (IEEE, 1998)
- Collision Sense Multiple Access/Collision Avoidance (CSMA/CA) – Controller Area Network CAN Bosh (1991)
- Token Passing – Token bus
- Mini Slotting ARINC 629
- Time Slot Allocation – Time Triggered Protocol (Kopetz, 1994), ARINC 659 (ARINC, 1993)

Two main types of data-buses are taking into account, TCP/IP and CANbus.Exist several variations based upon CANbus like FTT-CAN or planning scheduler. Based upon OSI computing layers protocols are defined (Tanenbaum, 2003).

Several aspects can be pursued such load balancing, scheduling analysis or synchronization. These are reviewed in this work following the basis of real-time and non-migration, using these basis load balancing is out the scope of this work, the interested reader may consult in Nara et al., (2003). For clock synchronization, there are various feasible approaches like Krishna et al., (1997) and Lönn (1999).

1.3.General Overview of TCP/IP Protocol

One of the mayor examples of this type of databuses is TCP/IP. The TCP/IP family protocols are defined to use the internet and other applications that use interconnected networks. The protocols are layered but do not conform precisely to the ISO 7-layer model. There are several layers used by TCP/IP as shown in Fig. 1.2.

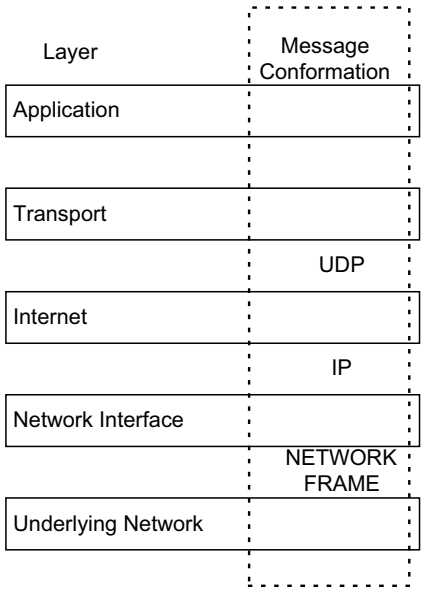

Fig. 1.2 TCP/IP Protocol over OSI Layer

The internet protocol layer provides two transport protocols TCP (Transport Control Protocol) and UDP (User Datagram Protocol). TCP is a reliable connection-oriented protocol and UDP is a datagram protocol that does not guarantee reliable transmission. The Internet Protocol (IP) is the underlying 'network' protocol of the internet virtual network.

TCP/IP specifications do not specify the layers below the internet datagram layer. The success of the TCP/IP protocols is based upon the independence of underlying transmission technology, enabling inter-networks to be built up from many single heterogeneous networks and data links.

1.4.Industrial Networks

It is very important in a distributed system to ensure system synchronisation. Without tight synchronisation it is likely that the system will lose data consistency. For example, sensors may be sampled at different times leading to failures being detected due to differences between data values. It is also important to consider intermediate data and consistency between replicated processing if comparison/voting is used to avoid the states of the replicas from diverging (Brasileiro et al., 1995). Asynchronous events and processing of non-identical messages could both lead to replica state divergence. Synchronisation at the level of processor

micro-instructions is logically the most straightforward way to achieve replica synchronism. In this approach, processors are driven by a common clock source, which guarantees that they execute the same step at each clock pulse. Outputs are evaluated by a (possibly replicated) hardware component at appropriate times. Asynchronous events must be distributed to the processors of a node through special circuits which ensure that all the correct processors will perceive such an event at the same point of their instruction flow. Since every correct processor of a node executes the same instruction flow, all the programs that run on the non-redundant version can be made to run, without any changes, on the node (as concurrent execution). There are, however, a few problems with the micro-instruction level approach to synchronisation. Firstly, as indicated before, individual processors must be built in such a way that they will have a deterministic behaviour at each clock pulse. Therefore, they will produce identical outputs. Secondly, the introduction of special circuits such as a reliable comparator/voter, a reliable clock, asynchronous event handlers, and bus interfaces, increases the complexity of the design, which in the extreme can lead to a reduction in the overall reliability of a node. Thirdly, every new microprocessor architecture requires a considerable re-design effort. Finally, because of their tight synchronisation, a transient fault is likely to affect the processors in an identical manner, thus making a node susceptible to common mode failures.

An alternative approach that tries to reduce the hardware level complexity associated with the approaches discussed above is to maintain replica synchronism at a higher level, for instance at the process, or task level by making use of appropriate software implemented-protocols. Such software-implemented nodes can offer several advantages over their hardware-implemented equivalents:

- Technology upgrades appear to be easy; since the principles behind the protocols do not change.
- Employing different types of processors within a node, there is a possibility that a measure of tolerance against design faults in processors can be obtained, without recourse to any specialised hardware.

Fail silent nodes are implemented at the higher software fault tolerance layer. The main goal is to detect faults inside of a number of processors (initially two) that compose a node. As soon as one of the processors has detected a fault it has two options; either remain fail silent or decrease its own performance. The latter option is suitable when the faulty processor is still checking information from the other processor. This implementation involves: firstly, a synchronisation technique called “order protocol” and secondly, a comparison procedure that validates and transmits the information or remains silent if there is a fault. The concept used for local fault tolerance in fail silent nodes is the basis of the approach followed in this thesis for the “smart” elements. However, in this case, in the presence of a fault the nodes should not remain silent.

The main advantage of fail silent nodes is the use of object oriented programming for synchronisation protocols to allow comparison of results from both processors at the same time. Fail silent nodes within fault tolerance are considered to be the first move towards mobile objects (Caughey et al., 1995). Although the latter technique is not explained here, it remains an interesting research area for fault tolerance.

System model and assumptions. It is necessary to assume that the computation performed by a process on a selected message is deterministic. This is the well-known assumption in state machine models for which the precise requirements for supporting replicated processing are known (Schneider, 1990). Basically, in the replicated version of a process, multiple input ports of the non-replicated process are merged into a single port and the replica selects the message at the head of its port queue for processing. So, if all the non-faulty replicas have identical states then they produce identical output messages. Having provided the queues with all correct replicas, they can be guaranteed to contain identical messages in identical order. Thus, replication of a process requires the following two conditions to be met:

Agreement: all the non-faulty replicas of a process receive identical input messages. *Order*: all the non-faulty replicas process the messages in an identical order.

Practical distributed programs often require some additional functionality such as using time-outs when they are waiting for messages. Time-outs and other asynchronous events, such as high priority messages, etc. are potential sources of non-determinism during input message selection, making such programs difficult to replicate. Further on (Chapter IV), this non-determinism is handled as an inherent characteristic of the system.

It is assumed that each processor of a fail-silent node has network interfaces for inter-node communication over networks. In addition, the processors of a node are internally connected by communication links for intra-node communication needed for the execution of the redundancy management protocols. The maximum intra-node communication delay over a link is known and bounded. If a non-faulty process of a neighbour processor sends a message, then the message will be received within δ time units. Communication channel failures will be categorised as processor failures.

1.5.Databuses

For Aerospace application it was first necessary to consider the databus standard to be used, on-engine for the distributed system. There are a number of standards used in aerospace. In the following sections the most common databuses are introduced.

ARINC 429
The ARINC 429 databus is a digital broadcast databus developed by the Airlines Electronics Engineering Committee's (AEEC) and Systems Architecture and Interfaces (SAI). The AEEC, which is sponsored by ARINC, released the first publication of the ARINC specification 429 in 1978.

The ARINC 429 databus (Avionics Communication, 1995) is a unidirectional type bus with only one transmitter. Transmission contention is thus not an issue. Another factor contributing to the simplicity of this protocol is that it was originally designed to handle "open loop" data transmission. In this mode, there is no required response from the receiver when it accepts a transmission from the sender. This databus uses a word length of 32 bits and two transmission

rates: *low speed*, which is defined as being in the range of 12 to 14.5 Kbits/s consistency with units for 1553b (Freer, 1989); and *high speed* which is 100 Kbits/s.

There are two modes of operation in the ARINC 429 bus protocol: character oriented mode and bit-oriented mode. Since the ARINC 429 bus is a broadcast bus, the transmitter on the bus uses no access protocols. Out of the 32-bit word length used, a typical usage of the bits would be as follows:

- Eight bits for the label
- Two bits for the source /Destination Identifier
- Twenty-one data bits
- One parity bit

This databus has the advantage of simplicity, however, if the user needs more complicated protocols or it is necessary to use a very complicated communication structure, the data bandwidth is used rapidly. One of the characteristics used by ARINC 429 is the LRU (Logical Remote Unit) to verify that the number of words expected match with those received. If the number of words does not match the expected number, the receiver notifies the transmitter within a specific amount of time.

Parity checks use one bit of the 32-bit ARINC 429 data word. Odd parity was chosen as the accepted scheme for ARINC 429 compatible LRU's. If a receiving LRU detects odd parity in a data word, it continues to process that word. If the LRU detects even parity, it ignores the data word.

ARINC 629
ARINC 629-2 (1991) has a speed of 2 MHz with two basic modes of protocol operation. One is the Basic Protocol (BP), where transmissions may be periodic or aperiodic. Transmission lengths are fairly constant but can vary somewhat without causing aperiodic operation if sufficient overhead is allowed. In the Combined Protocol (CP) mode transmissions are divided into three groups of scheduling:

- Level 1 is periodic data (highest priority)
- Level 2 is aperiodic data (mid-priority)
- Level 3 is aperiodic data (lowest priority)

In *level one* data is sent first, followed by *level two* and *level three*. Periodic data is sent in *level one* in a continuous stream until finished. Afterwards, there should be time available for transmission of aperiodic data. The operation of transferring data from one LRU to one or more other LRU's occurs as follows:

- The Terminal Controller (TC) retrieves 16-bit parallel data from the transmitting LRU's memory.
- The TC determines when to transmit, attaches the data to a label, converts the parallel data to serial data and sends it to the Serial Interface Module (SIM).

- The SIM converts the digital serial data into an analogue signal and sends them to the current mode coupler (CMC) via the stub (twisted pair cable).
- The CMC inductively couples the doublets onto the bus. At this point, the data is available to all other couplers on the bus.

This protocol has three conditions, which must be satisfied for proper operation: the occurrence of a Transmit Interval (*TI*), the occurrence of a Synchronisation Gap (*SG*), and the occurrence of a *TG* (Terminal Gap). The *TI* defines the minimum period that a user must wait to access the bus. It is set to the same value for all users. In the periodic mode, it defines the update rate of every bus user. The *SG* is also set to the same value for all users and is defined as a bus quiet time greater than the largest *TG* value. Every user is guaranteed bus access once every *TI* period. The *TG* is a bus quiet time, which corresponds to the unique address of a bus user. Once the number of users is known, the range of *TG* values can be assigned and the *SG* and *TI* values determined. *TI* is given by the following table.

Binary Value (BV)							BV	TI (ms)	TG (micro seconds)
TI6	TI5	TI4	TI3	TI2	TI1	TI0			
0	0	0	0	0	0	0	0	0.5005625	not used
0	0	0	0	0	0	1	1	1.0005625	not used
...	...	...	...	...	...	...	...	...	...
1	1	1	1	1	1	1	126	64.0005625	127.6875

Table 1.2 ARINC 629 time characteristics

To program the desired TG for each node, the user must follow Table 1.2 from *TI6* to *TI0* which represent the binary value (BV).

MIL-STD 1553b

Another commonly used databus is MIL-STD 1553b (Freer, 1989). This is a serial, time division multiplexed databus using screened twisted-pair cable to transmit data at 1Mbit/s. Data is transmitted in 16-bit words with a parity and a 3-bit period synchronisation signal, with a whole word taking 20 microseconds to be transmitted. Transformer-coupled base-band signalling with Manchester encoding is employed. Three types of devices may be attached to the databus:

- Bus Controller (BC)
- Remote Terminal (RT)
- Bus Monitor (BM)

The use of MIL-STD-1553b in military aircraft has simplified the specification of interfaces between avionics subsystems and goes a long way towards producing off-the-shelf interoperability. Most avionics applications of this databus require a duplicated, redundant bus cable and bus controller to ensure continued system operation in case of a single bus or controller failure. MIL-STD-1553b is intended primarily for systems with central intelligence and intelligent terminals in applications where the data flow patterns are predictable.

Information flow on the databus includes messages, which are formed from three types of words (command, data and status). The maximum amount of data which may be contained in a message is 32 data words, each word containing sixteen data bits, one parity bit and three synchronisation bits.

The bus controller only sends command words, their content and sequence determine which of the four possible data transfers must be undertaken:

- Point-to-Point between controller and remote terminal
- Point-to-Point between remote terminals
- Broadcast from controller
- Broadcast from a remote terminal

There are six formats for point-to-point transmissions:

- Controller to RT data transfer
- RT to controller data transfer
- RT to RT data transfer
- Mode command without a data word
- Mode command with data transmission
- Mode command with data word reception

and four broadcast transmission formats are specified:

- Controller to RT data transfer
- RT to RT(s) data transfer
- Mode command without a data word
- Mode command with a data word

This databus incorporates two main features for safety-critical systems, a predictable behaviour based upon its pooling protocol and the use of bus controllers. They permit communication handling to avoid collisions on the databus. MIL-STD-1553b also defines a procedure for issuing a bus control transfer to the next potential bus controller which can accept or reject control by using a bit in the returning status word.

From this information it can be concluded that MIL-STD 1553b is a very flexible data bus. A drawback, however, is that the use of a centralised bus controller reduces transmission speed as well as reliability.

Control Area Network Databus

This sort of data-bus is based upon CANbus. This type of databus is based upon the bottom two layers, its protocol is quite simple. This is based upon CSMA. This databus was defined by (Lawrenz, 1997). One of the key characteristics of this databus is (Kopetz, 1997). Other type of databus is MIL-STD1553 which is out the scope of work. This is a serial. Time division multiplexed databus using screened twisted-pair cable to transmit data at 1Mbit/s. Data is transmitted in 16 bit words with a parity and a 3-bit period synchronization signal, with a whole word taking 20 microseconds to be transmitted. Transformer-coupled base-band signaling with Manchester encoding is employed. Three types of devices may be attached to the databus:

- Bus Control
- Remote Terminal
- Bus Monitor

The use of MIL-STD- 1553b in military aircraft has simplified the specification of interfaces between avionics subsystems and goes a long way towards producing off-the-shelf interoperability. Most avionics applications of this databus require a duplicated, redundant bus cable and bus controller to ensure continued system operation in case of a single bus or controller failure. This databus is intended primarily for systems with central intelligence and intelligent terminals in applications where the data flow patterns are predictable.

Information flow on the databus includes messages, which are formed from three types of words (command, data and status). The maximum amount of data which may be contained in a message is 32 data words, each word containing sixteen data bits, one parity but and three synchronization bits. The bus controller only sends commands words, their content and sequence determine which of the four possible data transfers must be taken:

- Point-to-point between controller and remote terminal
- Point-to-point between remote terminals
- Broadcast from controller
- Broadcast from remote terminal

There are six formats for point-to-point transmissions:

- Controller to RT data transfer

- RT to controller data transfer
- RT to RT data transfer
- Mode command without a data word
- Mode command with data transmission
- Mode command with data word reception

This data bus incorporates two main features for safety-critical systems, a predictable behaviour based upon its pooling protocol and the use of bus controller. These permit communication handling to avoid collisions on the databus.

This databus has been model following several strategies such as Markov Chain and time delays as Nilsson (1998) has proposed. CANbus protocol is based upon of the protocol standard named Carrier Sense Multiple Access Collision Avoidance. A CAN word consists of six field as shown in Fig. 1.3.

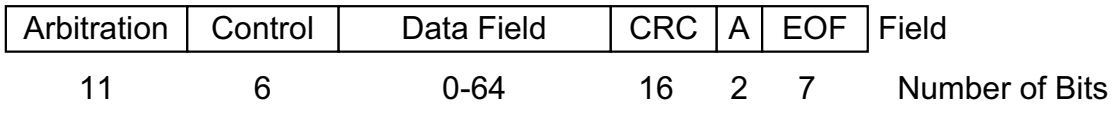

Fig. 1.3 Dataword Configuration from CANBUS

This databus is a broadcast bus where the data source may be transmitted periodically, sporadically or on-demand. The data source is assigned a unique identifier. The identifier serves as priority to the message. The use of this identifier is the most important characteristic of CAN regarding to real-time.

If a particular node needs to receive certain information then it indicates the identifier to the interface processor. Only messages with valid identifiers are received and presented.

The identifier field of a CAN message is used to control access to the bus after collisions by taking advantage of recessive bit strategy. For instance, if multiple stations are transmitting concurrently and one station transmits a '0' bit then all stations monitoring the bus see a '0'. When silence is detected each node begins to transmit the highest priority message held on its queue. If a node sends a recessive bit as part of the message identifier but monitors the bus and sees a dominant bits then a collision is detected. The node determines that the message it is transmitting it is not the highest priority in the system, stops transmitting and waits for the bus to become idle. It is important to recall that each message in CAN bus has a unique identifier which is based on the priority.

CAN, in fact, can resolve in a deterministic way any collision which could take place on the shared bus. When a collision occurs and arbitration procedure is set off which immediately

stops all the transmitting nodes, except for that one which is sending the message with the highest priority (lowest numerical identifier).

One of the perceived problems of CAN is the inability to bound the response time messages. From the observations above, the worst case time from queuing the highest priority message to the reception of that message can be calculated easily. The longest time a node must wait for the bus to become idle is the longest time to transmit a message. According to Tindell et al., (1995) the largest message (8 bytes) takes 130 microseconds to be transmitted.

The CAN specification (ISO 11898) discusses only the physical and data-link layer for a CAN network:

- The data link layer is the only layer that recognizes and understands the format of messages. This layer constructs the messages to be sent to the physical layer and decodes messages received from the physical layer. In CAN controllers, the data link layer is implemented in hardware. Because of its complexity and in common with most other networks this is divided into a:

    - Logical link control layer which handles transmission ad reception of data messages to and from other, higher level layers in the model.
    - Media Access control layer, which encodes and serializes messages for transmission and decodes received messages. The MAC also handles message prioritization (arbitration), error detection and access to the physical layer.

- The physical layer specifies the physical and electrical characteristics of the bus. This includes the hardware that converts the characters of a message into electrical signals fro transmitted messages and likewise the electrical signals into characters for received messages.

1.6.Concluding Remarks

A general review of some of the most common databuses is given specially a brief description of OSI layers and its relation to data communication through these databuses. This chapter gives an introduction of computer networks perform communications in order to understand the needs for real-time systems.

Chapter II

Real-Time Systems

2.1 Background

Nowadays Real Time Systems become a common issue in order to model computer systems behaviour in terms of time performance. Since the approach followed in this book is to present how computer communication affects control law performance, to achieve this strategy, it is necessary to understand how real time systems can be modeled and measured.

Several strategies conformed real time systems as a whole, these can be classified by two main aspects the needs and the algorithms of real time systems. First aspect allows to understand why real time is required over some conditions like the presence of the fault and the respective fault tolerance issue. Other like clock synchronization is necessary to review as need for real time systems in order to get a feasible communication performance.

On the other hand, a second aspect is related to how scheduling algorithms are focused into several aspects having an impact on system performance. This is reviewed in terms of consumption time and it is accomplished by time diagrams.

One of the most important issues on real time systems it is the conformation of time diagrams in order to define system behaviour under several scenarios. This strategy visualizes how the algorithm would perform on certain variations on time since this strategy gives the visualization of system response another issue arises related to how valid scheduling configuration it is, this is known as schedulability analysis.

Some other aspects such as load balancing, task precedence and synchronization are reviewed giving an integral overview of modeling real time systems and which are the repercussions of such an approach.

This revision of real time systems gives a strong idea of how control law is affected by these time variations which are the results of several conditions who are beyond the scope of this book. The important outcome of this review it is how time delays can be modeled in order to be defined under the control law strategy.

2.2 Overview

One of the main characteristics of real-time systems is the determinism (Cheng, 2002) in terms of time consumption. This goal is achieved through several algorithms that take into account several characteristics of tasks as well as the computer system where is going to be executed.

Real-Time systems are divided in two main approaches mono-processor and multiprocessor. These two are defined by different characteristics, first type has a common resource the processor and second approach has a common resource the communication link. Last common resource can be challenged through different communication approximations. For instance, the use of shared memory is common in high performance computing systems where the use of databus becomes common in network systems. Last approach (multi-processor) is the followed in this work.

There are two main sources of information worded to be review as introduction to real-time systems one made by Kopetz (1997) and other made by Krishna et al., (1997), both have a review of several basic concepts that are integrated in order to give a coherent overview of real-time systems like fault tolerance strategies, the most common protocols, those most common clock synchronization algorithms as well as some of the most useful performance measures. From this review one of the most important needs for real time systems is fault tolerance since its performance evaluation is modified in order to cover an abnormal situation.

Fault tolerance is a key issue, that has quite a lot implications in different fields such as the configuration in communication and the structural strategy to accommodate failures. Most of current strategies are based upon redundancy approach that can be implemented in three different ways:

- Hardware
- Software
- Time Redundancy

Hardware redundancy has a representation known as replication using voting algorithms named N-Modular Redundancy (NMR). Fig. 2.1 shows the basic structure of this type of approach. In this case several strategies can be pursued.

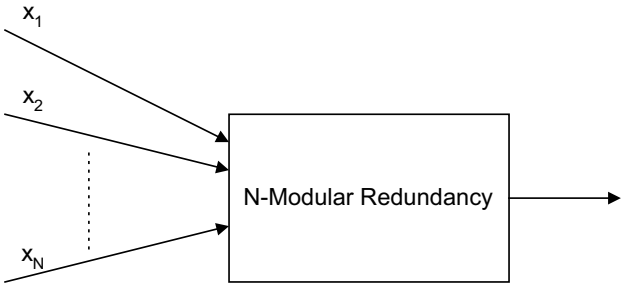

Fig. 2.1 N-Modular Redundancy Approach

Different approaches are defined as voting algorithms in order to mask faults in a trustable manner. These are classified in two main groups as safe and reliable algorithms. First group is referred to those algorithms that produce a safe value when there is no consensus between redundant measures. Alternatively, second group produce a value even in the case of no consensus, this last approach becomes quite common when safety is not an issue. Some of the most common voting algorithms are presented next:

- Majority Voter
- Weight Average Voter
- Median Voter

As example of safe algorithms, majority voter is presented: This algorithm defines its output as one of the elements of the largest group of inputs with the minimum difference. For instance, consider x_n inputs with a limit ε in order to evaluate the difference between two inputs $d(x_i,x_j)$. A group g is conformed by those inputs whose difference is lower than the limit ε. This voter can be defined as:

- The difference between two inputs is defined as $d(x_i,x_j)=|x_i-x_j|$
- Two inputs x_i and x_j belong to g_i if $d(x_i,x_j)<\varepsilon$
- The largest (in terms of the number of the elements) g_i is the winner and one of the elements that conform the group is the output of the voter.

As an example consider the next group of inputs {1.001, 1.0002, 1.1, 0.99, 0.98, 0.999} where the selected limit is $\varepsilon=0.01$. The difference between these elements is presented in Table 2.1.

Evaluated Values	**1.001**	**1.0002**	**1.1**	**0.99**	**0.98**	**0.999**
1.001	0	0.0008	0.099	0.011	0.021	0.002
1.0002	0.0008	0	0.0998	0.0102	0.0202	0.0012
1.1	0.099	0.0998	0	0.11	0.12	0.101
0.99	0.011	0.0102	0.11	0	0.01	0.009
0.98	0.021	0.0202	0.12	0.01	0	0.019
0.999	0.002	0.0012	0.101	0.009	0.019	0

Table 2.1Basic Table for Voting algorithm example.

From this table there are three groups $g_1=\{1.001, 1.0002\}$, $g_2=\{0.99, 0.98, 0.999\}$ and $g_3=\{1,1\}$ and the output of this voter is any of the elements of g_2 since is the largest group.

Another safety algorithm is the median voter, this algorithm selects the middle value from current group of inputs. In this case, the number of inputs has to be odd in order to select one single input. There are various ways to define this comparison, a common approach is the definition of differences between two input elements $d(x_i,x_j)$, considering x_n inputs. Where the difference between two inputs is defined as $d(x_i,x_j)=|x_i-x_j|$. The maximum difference value is discard and the two related values as well. This process is kept working until one element is left and declared as the output of the voter.

As example consider the same group presented in Table 2.1. From this group of elements, there is one drawback due to the number of elements is even, meaning, there is going to be one last pair of values which can be the output of the voter. In this case any of these values can be selected. The resultant Table is shown in Table 2.2 where those highlighted values as bold are the winners.

Evaluated Values	1.001	**1.0002**	1.1	0.99	0.98	**0.999**
1.001	*0*	*0.0008*	*0.099*	*0.011*	*0.021*	*0.002*
1.0002	*0.0008*	0	*0.0998*	*0.0102*	*0.0202*	**0.0012**
1.1	*0.099*	*0.0998*	*0*	*0.11*	*0.12*	*0.101*
0.99	*0.011*	*0.0102*	*0.11*	*0*	*0.01*	*0.009*
0.98	*0.021*	*0.0202*	*0.12*	*0.01*	*0*	*0.019*
0.999	*0.002*	**0.0012**	*0.101*	*0.009*	*0.019*	0

Table 2.2 Results of Median Voter evaluation based upon Table 2.1.

One example of a reliable algorithm is the weighted average algorithm. This algorithm (Lorczak, 1989) used the inputs x_i from *1* to *N* in order to produce an output based upon eqn. 2.1. In this case there are two values involved w_i and *s*. These two values are defined from eqns. 2.2 and 2.3.

$$x_o = \sum_{i=1}^{N} \left(\frac{w_i}{s} \right) x_i \tag{2.1}$$

$$s = \sum_{i=1}^{N} w_i \quad \text{for } i,j=1 \ldots N \text{ and } i \neq j \tag{2.2}$$

$$w_i = \frac{1}{\left(1 + \prod_{\substack{i=1, j=1 \\ i \neq j}}^{N} \left(\frac{d^2\left(x_i - x_j\right)}{\alpha^2} \right) \right)} \tag{2.3}$$

Where $d(x_i,x_j)$ is the difference between two inputs defined as $d(x_i,x_j)=|x_i-x_j|$. α value is a constant degree related to the sensibility of weights involved in eqn. 2.1.

From this kind of algorithms one concept of interconnection merges named fully connected system. This algorithm defines interconnection between all the involved components and a group of similar voting algorithms in order to reduce signal dependency as shown in Fig. 2.2 as well as masking local faults. Although the price to pay is an increase of communication time.

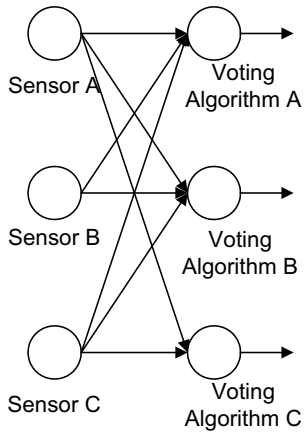

Fig. 2.2 Modular Redundancy Scheme

Alternatively, software redundancy is based upon masking software faults, which are quite different from hardware faults. These are not a consequence of certain conditions during operation. These are the result of design problems of the system. Redundancy becomes an open issue because there is no a proper definition of checking points to evaluate several software versions. In fact, the nature of the faults are defined in terms of design rather than exogenous effects of time malfunctions.

Certain strategies have been defined like n-version programming (Krishna et al., 1997) that are more related to how different programming teams interact in order to develop software rather than algorithms specifically designed for fault tolerance.

Another common approach is time redundancy defined through recovery points used to rolling back system execution when a fault is present. This corrective action takes place when a fault is present, then the system (or these elements that are affected by the fault) rolls back to a safe point before the failure has occurred. A similar approach is known as rolling forward strategy. In this case if a fault occurs those evolved elements go forward up to a safe point where it is known that the system has a fault free response.

An element that arises as result of this type of approximation is the evaluation of its performance. This issue becomes a mature topic by itself. Different approximations have been pursued for fault tolerance and real time. These are defined in terms of reliability, availability and time performance.

Reliability is defined (Kopetz, 1997) as the probability of a system that will provide certain valid response during a time window.

Availability has been defined as the probability that a system is performing correctly at an instant time (Johnson, 1989 and Johannessen, 2004). Other performance measures are defined in terms of time consumption and later responds.

Another important issue like fault tolerance is clock synchronization, several approaches can be pursued, although, some can be eliminated like current time adjustment as shown in Fig. 2.3.

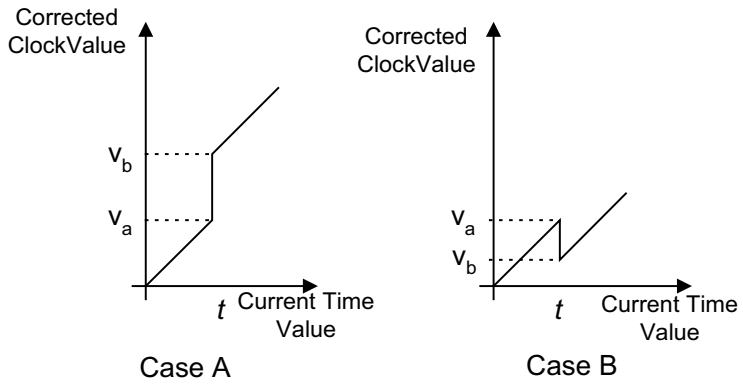

Fig. 2.3 Undesirable Time Correction

From both cases there is an un-desired correction. Case A at time *t* shows an un-correct clock value v_a which is corrected instantly to v_b value. This option is not valid due to abrupt forward clock modification and potential lost of current conditions. Similar situation is presented in case B where at time *t* there is an abrupt backward clock modification who is not acceptable due to lost of current conditions from one point to another.

In order to avoid this behaviour a common algorithm based upon clock skew can be followed using small changes between clocks. This approximation follows Fig. 2.4 where correction is performed using skew correction.

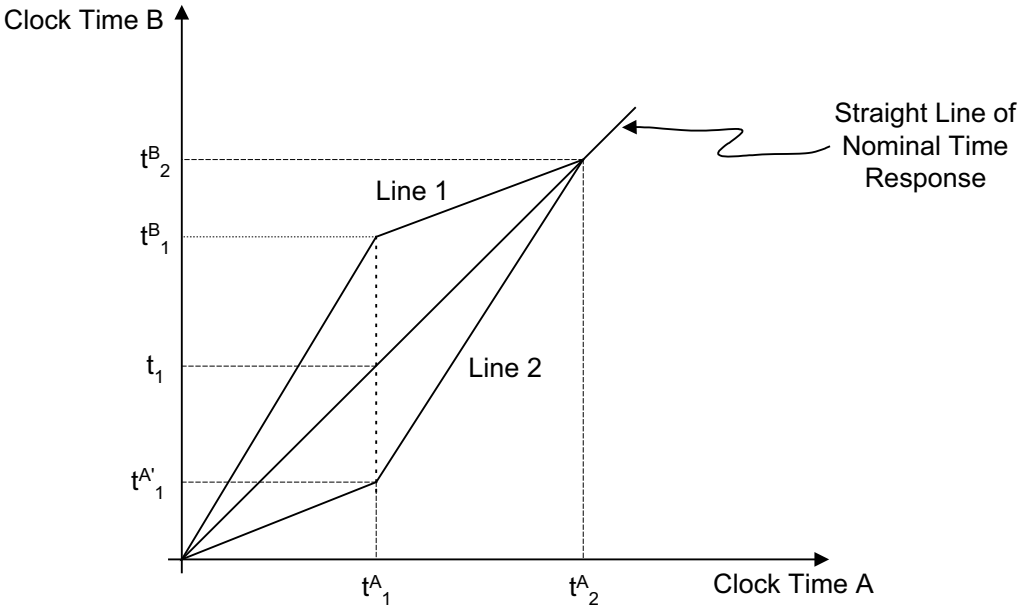

Fig. 2.4 Clock Skew

In this case, time correction is performed following gradual changes according to a nominal point in time named as (t^B_2, t^A_2). For instance, considering to be an observer in Line 2 at t^A_1 where there is a known difference with respect to a nominal time response between t_1 and $t^{A'}_1$.

This is corrected by the use of gradual clock modification until t^A_2 is achieved following eqn. 2.4.

$$clock\_Time\_B = \left(\frac{t_2^B - t_1^{A'}}{t_2^A - t_1^A} \right) * clock\_Time\_A \tag{2.4}$$

In this case time correction is obtained at t^A_2.

Another algorithm uses a similar principle but in a fault tolerance fashion where communication is presented and comparison between current available clocks takes place in each involved node. This is shown in Fig. 2.5. One disadvantage of this approach is related to communication vulnerability where time boundary should be present as shown in Fig. 2.6. If this boundary is lost by one of the nodes, the related clock missed its synchronization and it has to be performed consequently as broadcast manner. The result of this procedure is a communication overhead due time synchronization. Although it presents a reliable response against communication faults.

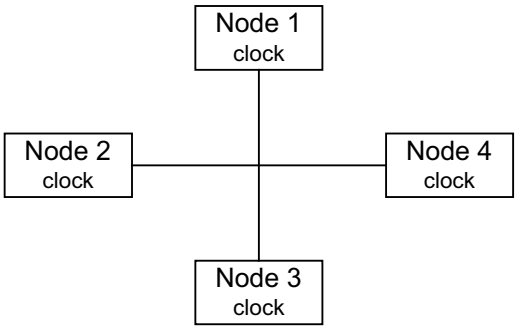

Fig. 2.5 Fault Tolerance Approach for Clock Synchronization

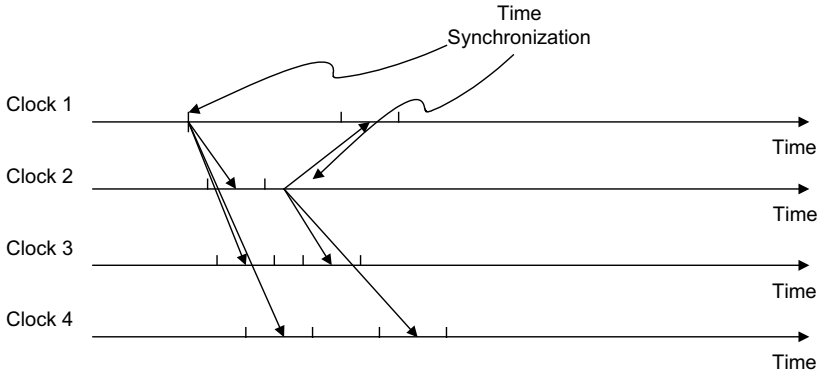

Fig. 2.6 Clock Synchronization with Bounded Time

Another strategy is known as bizantine clock (Krishna, 1997) which is common for intermittent faults (Lonn, 1999). Moreover, real time systems hold several characteristics that are compatible to other research areas like discrete control systems. To that respect real time control fundamentals have been explored by Törngren (1998) where the basics are established in common terms such as time delays and time variations in both areas (Table 2.3).

	Discrete Control Systems	Computing Systems
Activation	Time Triggered	Event Triggered
Time Delays	Commonly defined as Constant	Variable
Communication paradigm	Periodic Communication Strategy	Flexible Communication Strategy based upon Scheduling Algorithm
synchronization	Common Clock Synchronization, Sequential Procedure	Time Stamping Synchronization, Concurrent Programming
Time Variation	Bounded Time Variation	Bounded Time Variation with respect to Scheduling Algorithm.

Table 2.3 Common Characteristics between Computer and Control Systems

From these basics some common time intervals are defined like communication time, jitter, pre-processing time, events and so on. Based upon these representations a common graph is defined and named as time graph (Fig. 2.7), where the time behaviour of those components play a role into communication and need to be represented.

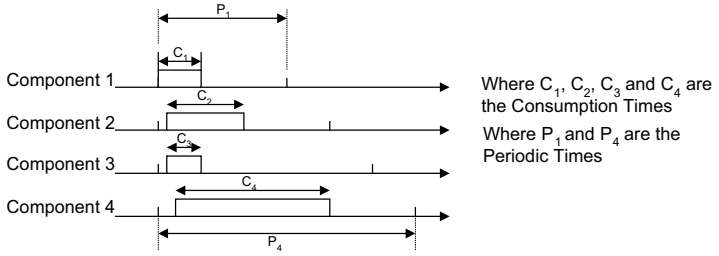

Fig. 2.7 Typical Time Graph

This Fig., presents the classical time graph between four components. This graph shows those necessary time intervals to bound timing behaviour from a real-time system. One element that plays an important role in communication is the jitter (*J*) that is defined as an uncertain time delay, which is a small fraction of any known time delay. It represents the undesirable variation of communication and computing times where it is not clear the cause of this. Fig. 2.8 shows a typical representation of the jitter between to elements during communication time.

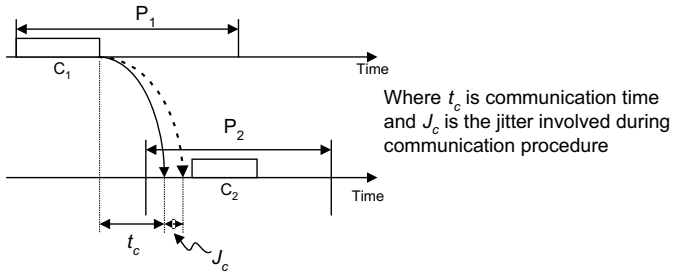

Fig. 2.8 Jitter Presence During communication Performance

From time graph representation, real time can be measured as an adding action from certain time intervals or events where c_1 is the consumption time of task 1, c_2 is the consumption time of task 2 and p_1 and p_2 are the related periodic times. For instance, Fig. 2.9 it is considered as an example, when an event occurs in component 1 and the respective flow chart follows the relation between components 1, 2 and 3, the consumed time (t_{tc}) from this procedure is defined as the sum of all elements involved in a consecutive transmission as shown in eqn. 2.5.

$$t_{tc} = t_{c1} + t_{ct1} + t_{pp1} + t_{c2} + t_{ct2} + t_{pp2} + t_{c3} \tag{2.5}$$

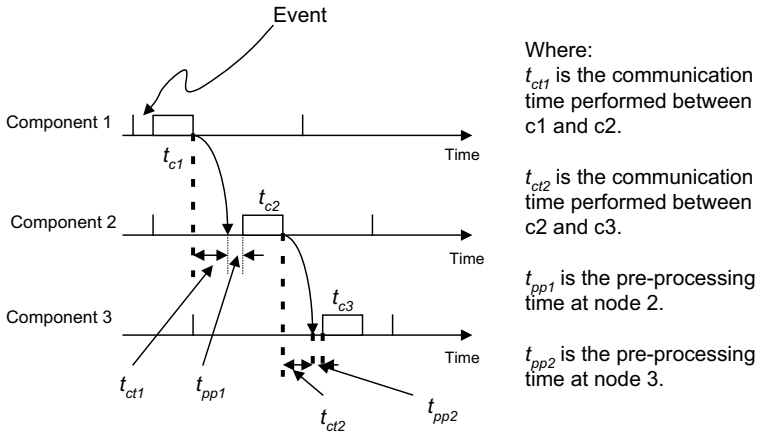

Fig. 2.9 Time Graph Describing Communication Procedure

In this case there is no presence of jitter behaviour which in real systems is uncommon. Nevertheless, since communication is bounded through this representation uncertainties are identified related to jitter assumption. Then, this measure becomes necessary to be known at least through the experience of ad-hoc equipment knowledge. This graph allows issues like complexity, mutual exclusion and up to certain extend load balancing as presented in future sections. An example of this strategy is a fault tolerance approach with clock synchronization as shown in Fig. 2.10.

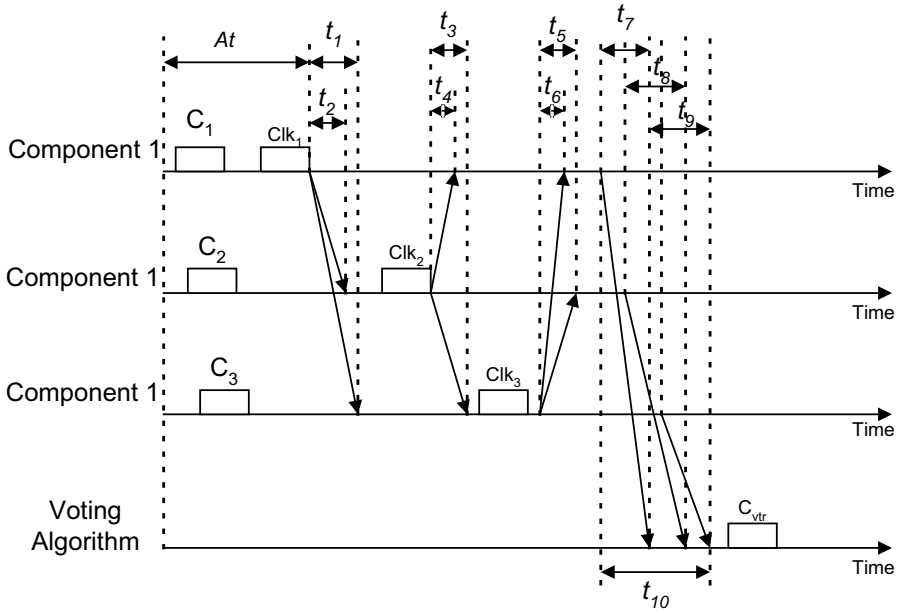

Fig. 2.10 Time Graph Representation of Fault Tolerance Approach

In this case communication times are t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8 and t_9. Consumption times are c_1, c_2 and c_3. Clock measurements are clk_1, clk_2 and clk_3. Consumption time related to voting algorithm is c_{vtr}. As the reader may realize these time intervals present an awkward characteristic related to the assumption of maximum communication times like t_1, t_3, t_5 and t_{10} due to a heuristic selection. This maximum final consumption time (T_{CT}) is presented in eqn. 2.5.

Having defined every component related to time graph. These maximum communication times are defined according to protocol and priority definitions.

$$T_{CT} = At + t_1 + Clk_2 + t_3 + Clk_3 + t_5 + t_{10} + C_{vtr} \tag{2.5}$$

This example presents two important cases like one processor sending messages to two different processors and three processors sending messages to one processor.

From this representation it is possible to define several characteristics such as the need of algorithms capable to define time behaviour and clock synchronization. To that respect, a class of algorithm named scheduling algorithm becomes essential.

Real-Time system is a multidisciplinary area related to modeling the behaviour of a system, to verify its behaviour and to analyze its performance. In order to review various strategies for modeling a real-time system Liu (2000) and Cheng (2002) have presented a good revision of several scheduling algorithms as well as formal representation such as deterministic finite state machines.

Moreover, Cheng (2002) presents several formal approximations to verify if a particular implementation to real-time systems is valid or not. In this direction, Koppenhoefer et al.,

(1996) has proposed a formal verification of distributed real-time control based upon periodic producer/consumer.

2.3 Scheduling Algorithms

The advantage of using scheduling algorithm into control systems allows to bound time delays as well as to define formal design of their effects into dynamic systems (Arzen, et al., 1999).

Scheduling algorithms allow to allocate tasks during certain time with respect to a common resource such as processor. These sort of algorithms are defined in terms of the common resource identified like processors and communication media. The most well known scheduling algorithms have been defined for first common resource where there are characteristics to be defined like scheduler analysis. For instance, scheduler analysis for mono-processor (the processor as common resource) approach is focused to be less than 1 where as scheduler analysis for multiprocessor approach (the communication as common resource) can be bigger than one according to the number of nodes to be involved.

At this section some of the most typical mono-processors algorithms are reviewed in order to view task allocation, performing for the case of multiprocessors algorithms these are mainly similar as their counterpart of mono-processors.

There are several algorithms that can be used such as RM (Rate Monotonic), EDF (Earliest Deadline First), FTT (Flexible Time Triggered, Almeida et al., 2002), LST (Least Slack Time). Where the difference between them is marked by the way tasks are ordered. It depends on the application which way of ordering tasks is the most suitable for a particular example. Those already mentioned algorithms are divided in two categories as static and dynamic schedulers. The main difference is that static scheduler defines during off-line process the allocation of task meanwhile dynamic scheduler allocates tasks based upon current conditions considering a time slot. For instance, consider three tasks with next characteristics (Table 2.3 and Fig. 2.11) under EDF algorithm if a task changes its deadline at Δt it would have a higher priority than those tasks already defined (Table 2.4).

	Consumption Time (C)	Periodic Time (P)	Deadline (D)	Priority
Task 1 (T_1)	C_1	P_1	D_1	Pr_2
Task 2 (T_2)	C_2	P_2	D_2	Pr_3
Task 3 (T_3)	C_3	P_3	D_3	Pr_1

Table 2.3 Tasks used to exemplified EDF Algorithm

From Table 2.3 task 3 has the smallest slack time (t_{s3}) therefore it has the highest priority Pr_1, thereafter, ask 1 has next priority and last task has the lowest priority Pr_3.

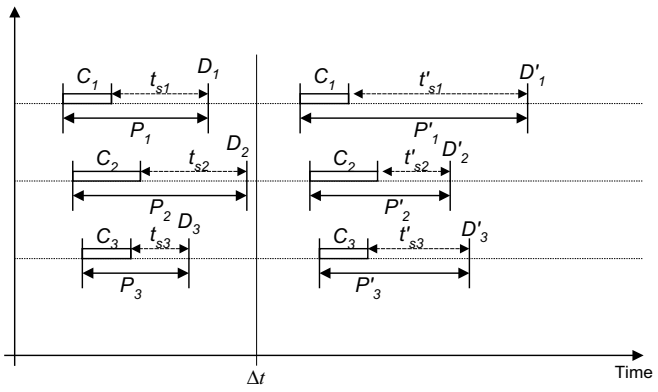

Fig. 2.11 Time Graph Related to Table 2.3

According to Fig., 2.11 there are two scenarios for these three tasks, firstly, task 1 has slack time ts_1, task 2 has slack time ts_2 and task 3 with slack time ts_3 giving the highest priority to task 3. Second scenario presents a different priority conformation according to slack times modifications.

	Consumption Time (C)	Periodic Time (P)	Deadline (D)	Priority
Task 1 (T_1)	C_1	P'_1	D'_1	Pr_3
Task 2 (T_2)	C_2	P'_2	D'_2	Pr_1
Task 3 (T_3)	C_3	P'_3	D'_3	Pr_2

Table 2.4 New Priority Order after At Reorganization

For the case of deadline modification as display in Fig., 2.11 priorities are modified as shown in Table 2.4 where task 2 has the smallest slack time (t_{s2}) therefore it has the highest priority Pr_1, task 3 has next priority and last task has the lowest priority Pr_3.

For real time purposes it is pursued to use static schedulers because its deterministic behaviour. Recently an approach named quasi-dynamic scheduling algorithms have been defined in order to give certain flexibility in static communication approach. An example of this sort of algorithm is the planning scheduler (Almeida et al., 1999). The planning scheduler is a pseudo-dynamic scheduler, in the sense that it presents some dynamic properties but is not fully dynamic. The underlying idea is to use the present knowledge about the system (in particular the variable set) to plan the system activity for a certain time window into the future. Such a time window is fixed, and independent of the periods of the variables, and is called a plan.

The scheduler must, then, be invoked once in each plan to build a static schedule that will describe the bus allocation for the next plan. The potential benefit of the planning scheduler in terms of run-time overhead is revealed by the following reasoning. Within a fixed time window of duration P_i like being the period of variable i among a set of N variables, there are at most S transactions

$$S = \sum_{i=1}^{N} \left(\left\lceil \frac{w}{P_i} \right\rceil + 1 \right) \tag{2.6}$$

Where idle time is manipulated in order to give an opportunity to sporadic tasks preemptable tasks to be expected. In order to perform task re-allocation macro-cycle of N tasks is divided in smaller windows named elementary cycles (EC) that are divided in basic units that are multiples of consumption times of every task. The only condition for an elementary cycle is that it has the same period as the fastest task. Since this partition is proposed the group of tasks conformed by *N* elements is re-organized according to these time restrictions taking into account periodic time sizes in order to define priorities of execution.

If any task does not have enough time to be executed during the respective EC with enough space for task performance. If there is one who is not able to fit in any EC it is said that this group of tasks is not schedulable.

Some other strategies for scheduling needs can be defined in a more ad-hoc manner from the basis of case study, for instance, some scheduling algorithms for control systems have been defined like Hong et al., (2002) and Hong (1995) where the approach is ad-hoc to the analyzed structure and named as bandwidth-scheduling algorithm. This algorithm proposes a timing analysis of each node time consumption (sensor, controller and actuator) considering data transmission time and the related time delays. Having established certain time boundaries and timing analysis of consumption time from every considered element, a review of the proposed algorithm is given. This algorithm consists of ordering elements such as sensors and actuators according to their inherent loop, for instance, sensors, controllers and actuators. Thereafter, this reordering is based upon earliest deadline first considering critique and non-critique zone from each node.

Each scenario has a correspondent control law considering some time delays conditions like communication time delays from sensor nodes, time consumption from several control nodes and those considered as sporadic time delays due to non-real time messages. Therefore, each modification established by the bandwidth scheduling algorithm has a proper repercussion into the dynamic modeling of the system and the respective controller. In this case, time delays are bounded and used to define control structure. This is reviewed in Chapter IV. The scheduling algorithm allows to define time delays boundary necessary for control law performance definition.

Alternatively, another ad-hoc scheduling algorithms have been proposed based upon fuzzy logic (Monfared et al., 2000). In this case an study of stochastic behaviour of process system is developed. A review of the stochastic nature from different scenarios allows the use of adaptive scheduling approach, although, it carries the respective uncertainty. This can be tackled by the use of a more restrictive adaptive approach, however, what is pursued by Monfared et al., (2000) is the use of fuzzy logic based upon the utilization of several membership functions to represents a poisson conditional probability function in order to adapt the best component configuration in terms of the manufacturing control system structure.

Another strategies focusing into hard and soft real-time communication using CANbus have been proposed by Livani et al., (1998) where the aim is to divide the message identifier from every CAN word into possible variants named hard real time and soft real time respectively. Messages get priority according to this classification. Accommodation of messages is according to high priority messages (hard real time messages) as pursued by EDF algorithm there are four main basic assumptions to be take into account:

- Each Real Time message has a reserved time slot.
- The reserved time-slot of each message is as long as the worst case transmission time of the message.
- The priority of a hard Real Time message depends on its transmission laxity.

An interesting approach of trade-off analysis of real-time control including scheduler analysis is proposed by Seto et al., (2001) where a review of optimal control based upon performance index defined as:

$$\left(\max_{u}\right)\left(\min_{u}\right)J(u) = \max\min\left[\, s\left(x\left(\left(t_f\right),t_f\right)\right) + \int_0^{t_f} L\left(x(t),u(t),t\right)dt \,\right] \tag{2.7}$$

Where *J(u)* is the performance index

S(.) and *L(.)* are the weight functions depending on systems states and control input.

t_f is the final time over the considered interval.

u(t) are the control inputs.

x(t) are the states function which are dependant on

$$\dot{x} = f\left(x(t),u(t),t\right) \tag{2.8}$$

and having as control input next function with related boundary.

$$c\left(x(t),u(t)\right) \le 0 \tag{2.9}$$

This function is minimized based upon the system dynamics and schedulability performance. As result of this optimization determination of the optimal frequencies for tasks schedule is performed by solving this nonlinear constrained optimization problem.

For instance Gill et al., (2001) have proposed an approximation for scheduling service based upon Real-Time CORBA middleware. This middleware strategy allows clients to invoke operation without concern of OS/Hardware platform, types of communication protocols, types of languages implementations, networks and buses (Vinoski, 1997). Specifically, the strategy pursued by Gill is a scheduling service that it has already implemented the most common scheduling algorithms from static and dynamic fashion. For instance, this service based upon a defined framework has already implemented RM, EDF, MUF (Maximum Urgency First) and

MLF (Maximum Laxity First) where an abstract implementation is followed based upon three main goals:

- Tasks dispatchments are organized by critical operation organized by a static priority where non-critical operations are dispatched by a dynamic scheduling.
- Any scheduling strategy must guarantee scheduling critical operations.
- Adaptive scheduling approach allows flexibility to adapt varying application requirements and platform requirements.

It also defines systems requirements by following a number of steps defined as:

1. Any application gives information used by TAO scheduling service (implemented as object) in order to define an IDL interface.
2. Time Configuration is performed either off-line or on.line as the application demands.
3. Scheduling service assigns static and dynamic priority.
4. Priorities assigned to each task and the respective sub-division to allow dispatching priority.
5. Schedulability is evaluated based upon priority assignment and the selected scheduling algorithm.
6. A number of queues necessary to dispatch the already ordered priorities (per node) are created per node.
7. Dispatching modules define thread priorities assignment according to previous analysis.

Following this idea of real time using middleware structure as resource manager, Brandt et al., (2002) have implemented a flexible real-time processing by developing a Dynamic Quality of service Manager (DQM) as a mechanism to operate on the collective quality of service level specifications. It analyzes the collective optimization of processes to determine its allocation strategy. Once the allocation is determined it defines the level that each application should operate in order to optimize global performance based upon soft real-time strategy where eventual mis-deadline is feasible. This approximation to real-time using middleware presents a competitive task allocation approach although dynamic management resource still compromise hard real-time behaviour for safety critical purposes.

Another approach related to real-time middleware is presented by Sanz et al., (2001), where an integrated strategy for several functional components like operational control is proposed. Such as Complex loops, sensors and actuators monitoring, planning execution and some others. This integration is proposed from the perspective of cooperative functional components encapsulated through agents using real-time CORBA. Furthermore, this author has explored the use of “intelligent” strategies for planning the behaviour of a complex network control system as presented in Sanz et al., (2003) where design patterns are reviewed in order to define the most suitable strategies for control law design, task allocation and exploit design knowledge.

Furthermore, the use of middleware strategies in order to define suitable scheduling algorithms has been explored by the use of CORBA (Sanz et al., 2001) . For instance Rate

Monotonic (RM) algorithm assigns priorities to tasks based on their periods: the shorter period the higher the priority. The rate of a task is the inverse of its period.

Rate monotonic algorithm behaves as following example in the presence of three tasks.

Example: Suppose we have three tasks with following characteristics.

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	A	a^T
T_2	B	b^T
T_3	C	c^T

Table 2.5 Three Tasks for Rate Monotonic Example

Where T_1 has the smallest period a^T and the smallest consumption time a. The related ordering of this table is presented in Fig. 2.12.

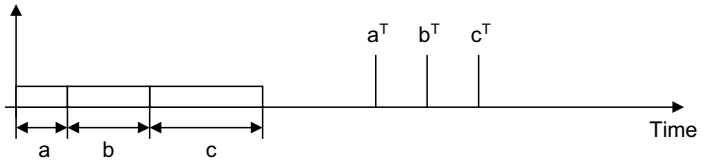

Fig. 2.12 Related order from Rate Monotonic Algorithm

Alternatively, in the case that we have periodic tasks with periods submultiples of bigger periods as shown in Table 2.6 where $2a^T = b^T$.

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	A	a^T
T_2	B	b^T
T_3	C	c^T
T_4	D	d^T

Table 2.6 Another Setting of task for Rate Monotonic Example

This group of tasks is organized following the basic principle of this scheduling algorithm with next distribution (Fig. 2.13).

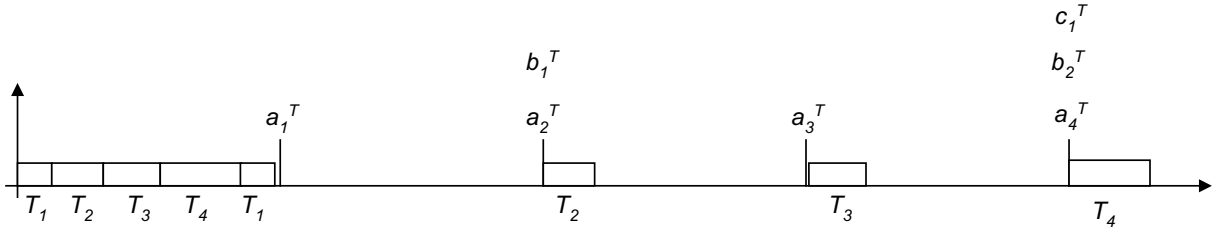

Fig. 2.13 Task Distribution from Table 2.6 According to Rate Monotonic

There are two main issues in this algorithm, firstly it has a common resource processor performance and secondly its organization allows a priory analysis in terms of the capacity to

allocate every task following their respective time restriction. This analysis is named schedulability analysis (Liu et al., 1973) where the basic condition is that the total percentage consumed by the consumed time (c_i) from every task with respect to its period (T_i) should be less than or equal to one. This condition is expressed as follows:

$$U = \sum_{i=1}^{N} \frac{c_i}{T_i} \leq 1 \tag{2.10}$$

Where N is the total number of tasks and U is named as the total percentage on consumption time. If this condition holds true, it is possible to reorganize this group of tasks as stated before.

There are various conditions to be reviewed around this algorithm. For instance, time variation with respect to time deadlines and consumption times where tasks can derive into this condition. Devillers et al., (2000) present a review of these variations where feasibility problem based on the utilization factor is possible or not.

On the other hand, The Earliest Deadline First (EDF) algorithm assigns priorities to individual jobs in the tasks according to their absolute deadlines. Earliest Deadline First (EDF) algorithm performs organization based upon the proximity of deadline with respect to current consumption time left from each task. It holds the scheduling analysis as rate monotonic algorithm. As an example Table 2.7 is presented.

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	A	a^T
T_2	B	b^T
T_3	C	c^T

Table 2.7 Task Distribution for EDF Algorithm

Where current time evaluation is pointed as c_t meaning time when is decided which task is going to be executed following EDF considerations (Fig. 2.14).

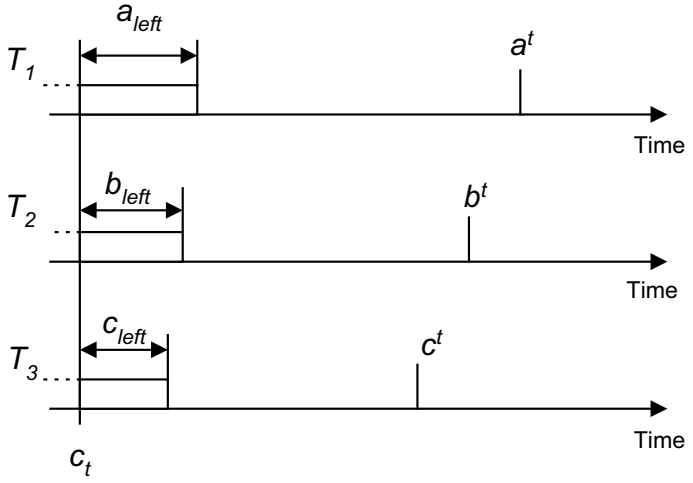

Fig. 2.14 Task Distribution for EDF Example

At c_t it is calculated the time left for task T_1 with respect to its own deadline as follows:

$$a^d = a^T - a_{left} \tag{2.11}$$

This procedure is performed for the rest of the tasks:

$$\begin{aligned} b^d &= b^T - b_{left} \\ c^d &= c^T - c_{left} \end{aligned} \tag{2.12}$$

Having produce a^d, b^d and c^d a comparison between this is performed:

$$\begin{aligned} a^b &< b^d \\ b^d &< c^d \end{aligned} \tag{2.13}$$

The smallest value from this comparison is the winner, therefore, it holds the capacity to use the common resource (Fig. 2.15) until condition (2.13) is modified.

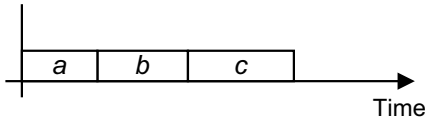

Fig. 2.15 Task Assignment to Common Resource According to EDF

The rest of the tasks are organized following the same criteria of deadline evaluation as shown in Fig. 2.15. If any task modifies either its consumption time or its deadline condition (eqn. 2.13) tasks re-arrange priorities and task execution is modified according to new conditions.

Alternative approach from dynamic scheduling strategy is the LST algorithm. Least Slack Time First (LST) algorithm at any time *t*, the slack of a job (t_s) with deadline at *d* is equal to *d-t* minus the time required to complete the remaining portion of the job *(Δt)* as shown in eqn. 2.14 and Fig. 2.16.

$$t_s = d - t - \Delta t \tag{2.14}$$

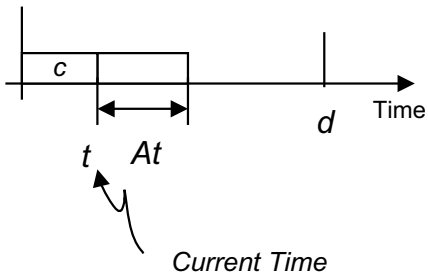

Fig. 2.16 Current Execution Task According to LST

As an example of this dynamic algorithm, Table 2.8 is proposed.

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	a	a^T
T_2	b	b^T
T_3	c	c^T

Table 2.8 Task Distribution for Related Example

Based upon Fig. 2.17 monitored system is performed by current time t as follows:

$$\begin{aligned} t_{sa} &= a^T - \Delta t_a - t \\ t_{sb} &= b^T - \Delta t_b - t \\ t_{sc} &= c^T - \Delta t_c - t \end{aligned} \tag{2.15}$$

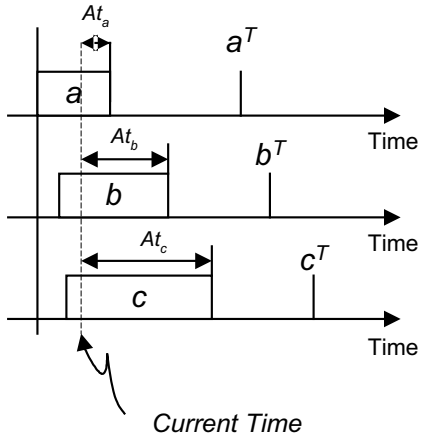

Fig. 2.17 Time Evaluation According to LST

Where t_{sa}, t_{sb} and t_{sc} are the respective slack time of each task priority, in this example the task who win the common resource is a because t_{sa} is the smallest value then it has the biggest priority. This algorithm has the particularity that behaves as EDF algorithm according to certain conditions.

Another dynamic scheduling algorithm is the maximum urgency first (MUF) algorithm. This algorithm organizes a group of task following next procedure, it follows EDF procedure combining a heuristic priority designation of tasks when both techniques agreed to certain priority assignment then selected task is performed.

It takes into account deadline proximity as EDF algorithm and priority assignment based upon exogenous demands from case study. As example of this algorithm Table 2.9 is proposed.

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	a	a^T

T_2	b	b^T
T_3	c	c^T

Table 2.9 Task Distribution for MUF Algorithm Priorities

Where task organization is presented in Fig. 2.18.

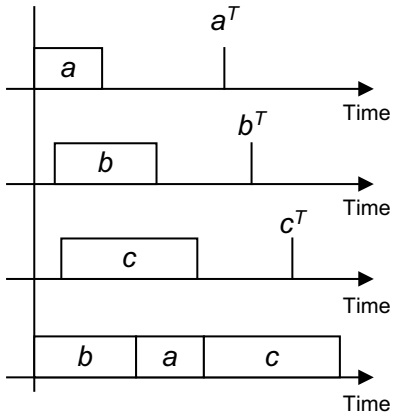

Fig. 2.18 Task Original Organization from Table 2.9

If we check out the result of this re-ordering (last part of Fig. 2.18) it has been performed in a very obvious strategy since exogenous priority have won according to Table 2.9.

As concluding remark from this section, different scheduling algorithms have been presented as well other strategies such as ad-hoc scheduling algorithms responses as well as tasks organization viewing those key advantages and disadvantages of different algorithms. One procedure that can be followed to define a feasible approach is presented in next section. This review gives an idea of how scheduling approach can be followed taking into account performance or middleware strategies.

Basically, the protocol makes use of the dual-phase elementary cycle concept in order to combine time- and event-triggered communication with temporal isolation. Moreover, the time-triggered traffic is scheduled online and centrally in a particular node called master. This feature facilitates the online admission control of dynamic requests for periodic communication because the respective requirements are held centrally in just one local table.

2.4 Distributed Real-Time Systems

Various factors should be taken into account in order to define a scheduling algorithm for distributed systems. Issues like synchronization arise as fundamentals for this purpose other characteristics are listed next:

- Synchronization
- Communication Cost

- Load Balancing

- Task Assignment and task precedence.

Time synchronization has been reviewed in previous section defining the most common algorithms like time stamping, passing time synchronization and sliding linear regression as checked by Johannessen (2004). Techniques like passing-time synchronization have as main characteristic the use of inherent protocol with time managing like network time protocol. Cervin et al., (2003) has arise the issue of synchronization where synchronization clocks by subnets organizations is commonly pursued although it has a high timing cost that affects performance by inherent time delays.

From this group of possible strategies arise the issue of performance evaluation in order to determine a suitable approach for certain cases of study, for instance, Lönn (1999) presents review of different performance evaluation techniques as well as some results with respect to a specific configuration such as fault tolerance average. Where this approximation presents the best results in average skew and mean time between faults.

This strategy (Kopetz, et al., 1987) is classified as converge clock, it consists of the average of all n-clocks except the *n* fastest clock and the *m* slowest clock. Maximum skew is presented in eqn. 2.16.

$$\partial_{\max} = (\varepsilon + 2\rho R)\frac{n-2m}{n-3m} \tag{2.16}$$

where ε is the reading error of a remote clock, R is the re-synchronization interval ρ the maximum drift between two clocks, n and m is ∂ from the result of this eqn. (2.16). Clocks are corrected in terms of this error named ∂_{max}.

On the other hand, communication cost is defined as the rate between the size of the data to be transmitted and the frequency of transmission. This cost can be defined as percentage between these two values, although it is based upon the characteristics of case study. For instance, some distributed systems can be loosely connected, therefore, transmit with a very low frequency but with high loaded data in terms of information such as reviewed by Coulouris et al., (1994). Load balancing is performed when there are various processes and their respective processors therefore accommodation is performed by several algorithms that take into account performance measures from both, processes and processors. The load balancing algorithm is an ad-hoc approach to case study, where the key factor is how to define performance in terms of the analyzed variables.

There are restrictions related to computational activities where several processes can not be executed in arbitrary order but have to take into account precedence relations defined when design stage take place (Buttazzo, 2004). These relations are described through a-cyclic graph, where tasks are represented by nodes and precedence relations by arrows (Fig. 2.19).

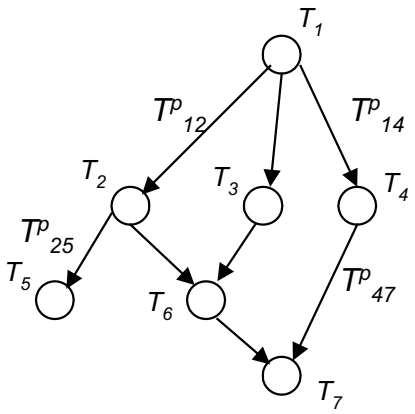

Fig. 2.19 Task Precedence Example

Where $\{T_1, T_2, ..., T_7\}$ are the nodes and $\{T^P_{14}, T^P_{12}, ..., T^P_{47}\}$ are the related precedence tasks. Task precedence is defined by case study and it becomes a requirement for scheduling algorithm.

There are other implementations like that proposed by Altisen et al., (2002) where a compromise between scheduling algorithm and control synthesis paradigm is proposed

Since the scope of this work is related to distributed system the use of those pieces allow to cover the main picture of real-time distributed systems which is to define how time delays can be modeled up to certain conditions like consumption time defined by total time spent from a group of tasks organized by a particular scheduler algorithm. Therefore an important issue to arise is related to time snchronization between processors, this is performed by various means as shown through this chapter. After this review of different algorithms the proposal of specific strategy is defined in terms of the needs of case study as reviewed in Chapter 6.

2.5 Conclusions

As concluding remark of this chapter is the brief overview of real-time systems and various components that take part in this review such as fault tolerance clock synchronization and scheduling algorithms. It has been reviewed new paradigms into that respect like the use of middleware in real time and the directions of this strategy in the near future.

Chapter III

SMART PERIPHERAL ELEMENTS

3.1 Overview

The main characteristics of this sort of peripheral elements are constituted by communication capabilities, fault diagnosis and certain degree of autonomy. This idea has been explored in terms of smart networks (Reza, 1994).

Different research groups have explored an interesting review of this sort of configured element. One of the key of holistic definitions of this technology has been made by Masten (1997). What is expected from this strategy is to define autonomous elements capable to detect faults in order to take fault tolerance actions like structural reconfiguration as previous step for control reconfiguration. This chapter is focus in several strategies in order to enhance fault detection and localization capabilities to peripheral elements under the prevalence of smart elements.

3.2 Peripheral Autonomy

Peripheral autonomy is defined by the capability to produce results even in fault conditions. Fault diagnosis is a mature defined area where several approaches to detect, isolate and diagnosis a fault have been defined (Patton et al., 2000). For instance, knowledge based approach using neural networks is a feasible option as presented by Chiang et al., (2001). Other approaches like signal analysis based are feasible as has been presented by Campbell et al., (2004) and Gertler (1998). As mention on previous chapters stability based approaches such as robust estimation base have been studied by Mangoubi (2000) and Chen et al., (1999) where uncertainty becomes an issue during fault presence. A complete survey of recent develop algorithms is presented by Venkatasubramanian et al., (2003a, 2003b and 2003c) looking at all kind of classical strategies from model based to model free techniques.

The strategy for peripheral autonomy opens a new area of work known as smart sensor networks that assigns new challenges such as multivariate pattern recognition and cooperative networking as presented by Agre et al., (1999), Reza (1994) and Akbaryan et al., (2001). Studies for sensor networks have been reviewed at different perspectives such as fault diagnosis and real time. Since the information obtained from this configuration allow the use of data fusion, fault tolerance, structural reconfiguration, control reconfiguration and other

techniques for keeping performance up to certain levels one of the most important features in here is how to evaluate sensor network configuration and the afterward technique like those already mentioned. For this respect, different views are reviewed from a local to global point of views. First view is a boarded in this chapter by defining a heuristic measure named as confidence value. Meanwhile global view is revised in chapter 4 as part of global evaluation from the impact of peripheral autonomy into system performance.

Since peripheral autonomy is one of the main advantages from using a smart peripheral element and certainly is the main reason for using it as central aspect from this kind of elements on this book, it is necessary to define how to accomplish autonomy in terms of fault detection. Following section on this chapter are focused into this goal.

3.3 Typical Smart Elements

A "smart" element is defined as a device that can communicate, self-diagnose and make decisions (Masten, 1997). Based upon this definition a "smart" element (SE) can be visualised as shown in Fig. 3.1. The main goal of the device is to obtain as much information as possible in order to produce self-calibration and compensation based upon structural analysis (Blanke et al., 2003). Additionally, this information must be processed and packaged in a standard way to be transmitted over the communication network supported by the distributed system.

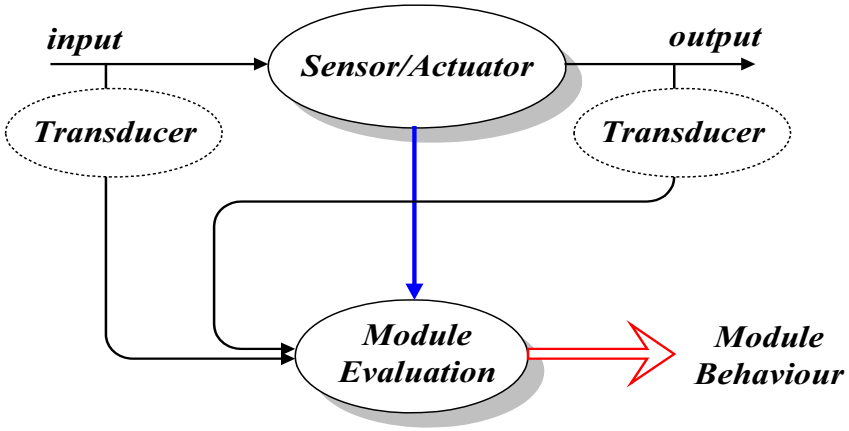

Fig. 3.1 "Smart" model

For the purpose of this work, "smart" elements consider just two kinds of peripheral devices: sensors and actuators.

These devices play the role of independent elements for the distributed system (Fig. 3.2). Together, with the controller they must perform their tasks within the restrictions on time dictated by the scheduler. However, in the presence of abnormal conditions the overall system must be robust to deal with any delay caused by either the fault or the accommodation procedure. In this work to measure the impact in terms of time degradation of these procedures a simulated distributed system is utilized. This is explained in chapter V.

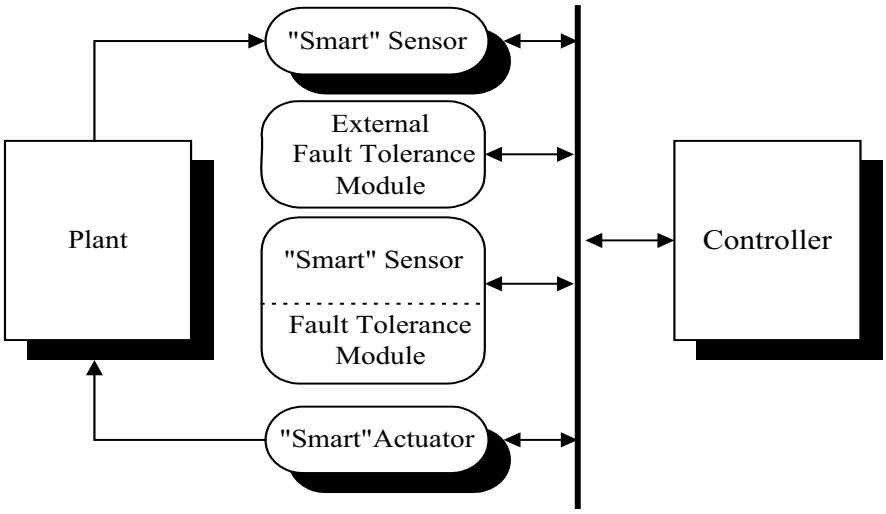

Fig. 3.2 Network Concept

Fig. 3.2 shows different approaches to “smart” sensors combined with local fault tolerance strategies. A “smart” sensor may rely on an external module for fault tolerance or it may have in-built fault tolerance. Similarly, actuators may adopt either of these approaches.

Technological progress in microelectronics and digital communications has enabled the emergence of “smart” or “intelligent” elements (devices with internal processing capability). Conceptually, these devices can be divided into the transducer and the transmitter parts, which are integrated in one unit. Moreover, the decentralisation of intelligence within the system and the capability of digital communications makes it possible for “smart” elements to yield measurements of better quality (Ferree, 1991) due to better signal processing, improved diagnostics and control of the local hardware.

“Smart” sensors and actuators are developed to fit the specific requirements of the application. However, consistent characteristics have been defined by Masten, (1997) for smart sensors and actuators. This standard defines a “smart” element as a device, which has the capabilities of self-diagnosis, communication and compensation on-line.

In particular, “Intelligent” sensors offer many advantages over their counterparts, e.g. capability to obtain more information, produce better measurements, reduce dependency and increase flexibility of data processing for real-time. However, standards need to be developed to deal with the increased information available to allow sensors to be easily integrated into systems. The adoption of the Fieldbus standard for digital communications allows the sensor to be treated as a richer information source (Yang et al., 1997a).

Nowadays, modular design concepts are beginning to generate specifications for distributed control. In particular, systems are appearing where low level sensor data is processed at the sensing site and a central control manages information rather than raw data (Olbrich et al., 1996a). In addition, process control is becoming more demanding, catalysing demands for improved measurement accuracy, tighter control of tolerances and further increases in automation (Olbrich et al., 1996b). The degree of automation and reliability that is likely to be

required in each module will almost certainly demand high sensitivities, self-calibration and compensation of non-linearities, low-operation, digital, pre-processed outputs, self-checking and diagnostic modes. These features can all be built into "smart" sensors.

Likewise, low cost microelectronics allows integration of increased functionality into distributed components such as actuators. This has led to the rise of mechatronics as an interesting new research field. Here, electronic control is applied to mechanical systems using microcomputers (Auslander, 1996). Using a microprocessor it is possible to program an actuator to perform a number of additional functions resulting in a number of benefits (Masten, 1997):

- Automatic actuator calibration
- Lower cost installation
- Preventive maintenance reduction
- On-site data collection

The high capabilities of microelectronics allow new features to be integrated together for fault detection and isolation. "Smart" elements are becoming more widespread (Isermann, 1994). The most common actuators transform electrical inputs into mechanical outputs such as position, force, angle or torque. For actuators, the classification and evaluation can be concentrated into one of three major groups:

- Electromechanical actuators
- Fluid power actuators
- Alternative actuator concepts

In the future, further development of actuators (Raab and Isermann, 1990) will be determined by the following general requirements:

- Greater reliability and availability
- Higher precision of positioning
- Faster positioning without overshoot
- Simpler and cheaper manufacturing.

Below, the different modules of the information flow of a 'low-degree intelligent actuator' (Isermann and Raab, 1993) are given. They comprise of these particular requirements:

- Control at different levels
- Self-tuning/adaptive (non-linear) control
- Optimisation of the dynamic performance
- Supervision and fault diagnosis
- Knowledge base

Analytical knowledge:

- Parameter and state estimation (actuator models)

- Controller design methods

Heuristic knowledge:

- Normal features (storage of learned behaviour)
- Inference mechanism
- Decisions for (adaptive) control
- Decisions for fault diagnosis
- Communication

Internal: connecting of modules, messages
External: with other actuators and the automation system.

Hence, the 'intelligent' actuator adapts its internal controller to the non-linear behaviour (adaptation) and stores its controller parameters dependent on the position and load (learning), supervises all relevant elements and performs a fault diagnosis (supervision) to request for maintenance. If a failure occurs, it can be configured to fail-safe (decisions on actions) (Isermann and Raab, 1993).

Focusing on "smart" actuators, Koenig et al., (1997) proposed a FDI algorithm based upon the idea of hierarchical detection observers (Janseen and Frank, 1984) to enable detection and isolation of a large variety of faults for a system under real-time computation constraints.

An example of FDI applied to induction motors is presented by (Beilharz et al., 1997) using a parameter estimation technique. The novelty of this approach is in the calculation of the parameters based upon the supplied signals with different frequencies. Moreover, Lapeyre, (1997) proposed an on-line parameter estimation based on the modified version of the extended Kalman filter (Ljung, 1979). A similar approach for FDI is proposed by Oehler et al., (1997) using extended Kalman filters to make the parameter estimation possible. Furthermore, (Benchaib et al., 1997) proposes a particular type of observer named the self-tuning sliding mode observer (Kubota, et al., 1993) to detect faults in a specific type of induction motor. Mediavilla et al., (1997) propose parity equations for multiplicative faults (as described by Gertler et al., (1995)) focused on an industrial actuator benchmark designed by Blanke et al. (1994).

3.4 Smart Elements Designs

To design a peripheral element based upon the concept of fault diagnosis it is necessary to define the monitored element in terms of structural analysis in order to determine which kind of faults are detectable. Since this available information is present suitable FDI strategy can be defined in terms of available faults. Structural analysis allows system modeling in terms of monitorability with the hand of petri nets fashion.

Firstly, fault dynamics of monitored elements are defined, secondly structural analysis is defined as petri nets, finally FDI is pursued. Additionally, the use of neural networks or fuzzy

logic can be challenged ion order to determine behaviour of system during the presence of faults.

3.5 Fault Diagnosis Approximations

Since the basic issue in smart element design is the fault diagnosis procedure, it is necessary to give a review of those available strategies to this respect. Fault detection and isolation is divided in two main groups, qualitative and quantitative approximations, both contain several algorithms presented in Table 3.1.

Qualitative Methods	**Quantitative Methods**
Statistical Methods (PCA)	Parameter Estimation
Neural Networks	Unknown Input Observers
Fuzzy Logic	Parity Equations
Causal Models	Kalman Filters

Table 3.1 Most Common Strategies for Fault Detection and Isolation

Each algorithm present certain advantages and disadvantages to characteristics of the process to be mentioned. In order to give a fair comparison of these algorithms a brief review of them is performed.

3.5.1 Parameter Estimation

Parameter estimation is based upon static model identification where the basic representation is given by

$$y(t) = a_1 u_1(t) + ... + a_N u_N(t) \tag{3.1}$$

Where two vectors are conformed, a regression vector $U = [u_1(t)..u_N(t)]$ and a parameter vector $\theta = [a_1,...,a_N]$ giving next representation

$$y(t) = U^T \theta \tag{3.2}$$

Now this output can be predicted based upon an estimation of parameter vector like follows

$$\hat{y} = U^T \hat{\theta} \tag{3.3}$$

Therefore the related performance index is defined as:

$$J = \sum_{i=0}^{N} [\hat{y}(t-i) - y(t-i)]^2 \tag{3.4}$$

where N is the number of elements.
Using this performance index J and its derivative in order to define the optimum of $\hat{\theta}$ value. From this evaluation $\hat{\theta}$ is defined as:

$$\hat{\theta} = \left[\sum_{i=0}^{N}\left(u(t-i)u(t-i)^T\right)\right]^T\left[\sum_{i=0}^{N}\left(u(t-i)y(t-i)^T\right)\right] \tag{3.5}$$

Or in other words

$$\hat{\theta} = \left[U^T U\right]^T U^T y \tag{3.6}$$

For the case of non linear dynamic models parameter estimation can be defined in terms of classical models, named as moving average (MA) and autoregressive moving average (ARMA). There are different approaches from the number of input-output variables such as single input single output system multiple-input single-output system and multiple-input multiple-output system. For any case two kinds of faults can be detected additive and multiplicative. For the case of additive faults these are considered as exogenous effects into peripheral elements such as, sensors or actuators. Mathematical representation of this effect is given in next eqn.

$$\begin{aligned} x(k+1) &= Ax(k) + B(u(k) + \Delta u(k)) \\ y(k) + \Delta y(k) &\cong Cx(k) \end{aligned} \tag{3.7}$$

where Δy and Δu are the related variation known as additive faults x and y are the states and output respectively. A, B and C are well dimension and known matrices. For the case of multiplicative faults these are presented into the monitored element, therefore, the representative matrix would suffer modifications like the follows:

$$\begin{aligned} x(k+1) &= (A + \Delta A)x(k) + B(u(k)) \\ y(k) &= Cx(k) \end{aligned} \tag{3.8}$$

Where ΔA represents the inherent variations of the system named as multiplicative faults. From this fault representation, these are reviewed based upon residual generation which are categorized in two types structural and directional (Gertler, 1998). Structural residuals are designed that each variable corresponds to a specific subset of faults as shown in Fig. 3.3, where r_1, r_2 and r_3 are residuals produced by any model based technique like tipically observed based technique. Fault_1, fault_2 and fault_3 are faults related to residual coordination when they act over the monitored system.

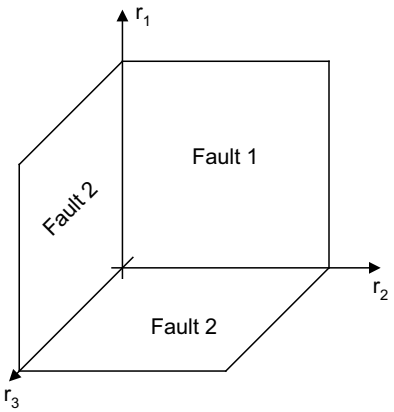

Fig. 3.3 Structural Fault Decomposition

Directional Residuals are designed to response to a particular fault as shown in Fig. 3.4. In this case, faults response to a particular behaviour of three residuals giving a resultant response considering a direction of certain behaviour. This sort of strategy tends to be quite useful if fault response is very well known in order to define certain replay boundaries.

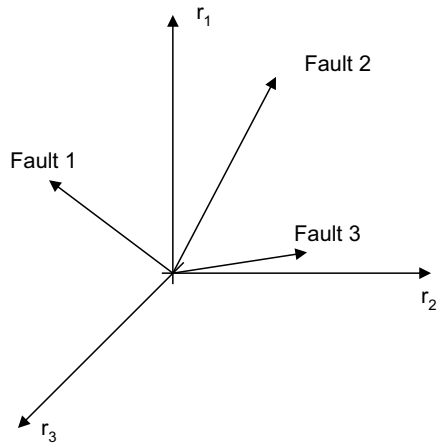

Fig. 3.4 Directional Fault Decomposition

3.5.2 Observer Based Techniques

Another common strategy for detection is the named state space fault detection. This is based upon next eqn.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{3.9}$$

this procedure is based upon state observers which are used to reconstruct the un-measurable state variables following general eqn. 3.10 are

$$\begin{aligned} \hat{x}(k+1) &= A\hat{x}(k) + Bu(k) + He(k) \\ e(k) &= y(k) - C\hat{x}(k) \end{aligned} \tag{3.10}$$

Where H is the gain observer matrix and e is named as error. Now the state estimation error follows eqn. 3.11.

$$\begin{aligned} \tilde{x}(k) &= x(k) - \hat{x}(k) \\ \tilde{x}(k+1) &= [A - HC]\tilde{x}(k) \end{aligned} \tag{3.11}$$

where H is gain matrix related to the state observer and $\tilde{x}$ the state vector error. There are two possible approaches, firstly observer can be sensitive to fault presence by proper model of H matrix. Second, approach is complete the opposite where observer does not response to fault. Then residual becomes necessary as monitored variable in order to detect faults.

3.5.3 Parity Equations

Other common methodology is named parity equations, this is based on a description the model using a linear process representation through a transfer function like expressed in eqn. 3.12.

$$H_P = \frac{y}{u} = \frac{B}{A} \tag{3.12}$$

where B and A are algebraic structures that represents the monitored element.

This process can be represented through a similar structure named process model

$$H_m = \frac{y_m}{u_m} = \frac{B_m}{A_m} \tag{3.13}$$

now if a f_u is a fault added to the input and f_y is a fault added at the output, the process can be represented as follows.

$$y = H_P u + H_P f_u + f_y \tag{3.14}$$

where the related error is given by:

$$e = y - y_m = H_p f_u + f_y \tag{3.15}$$

Following typical configuration shown in Fig. 3.5

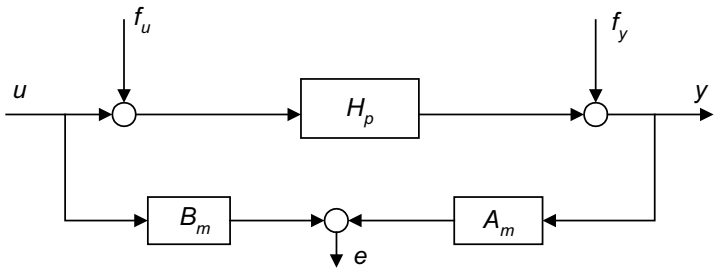

Fig. 3.5 Basic Configuration of Parity Equations.

The error allows detection since there is a difference either at input or output of monitored element.

3.5.4 Principal Components Analysis (PCA)

Another strategy is based upon statistical modelling like PCA. The PCA technique is a linear technique of multivariate analysis that aims to reach a linear and orthonormal special transformation as

$$y = Wx \tag{3.16}$$

where

$$x = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}^T$$

is a standardized input vector,

$$y = \begin{bmatrix} y_1 & y_2 & y_3 & \dots & y_m \end{bmatrix}$$

are the principal components and

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{mn} \end{bmatrix} \in R^{mxn}$$

is the transformation matrix whose components are called principal vectors or directors (Misra et al., 2002, Jolliffe, 2002).

The aim of this technique is to minimize the error when $x(t)_i$ is approximated using $k(\angle n)$ components of $y : \hat{X} = \sum_{i=1}^{k} w_i^T y_i$ this approach produces an error $E = |x - \hat{x}_i|$. This orthonormal transformation, *W*, is obtained by the eigenvectors of the correlation matrix of *x(t)* and the error as a function of their eigenvalues (Moya et al., 2001).

This projection matrix as in orthogonal projection is performed as shown in two dimensions (Fig. 3.6). In here *x* vector is composed of two dimensions that is projected to vector *y* where the difference between these two vectors is named as *r* vector.

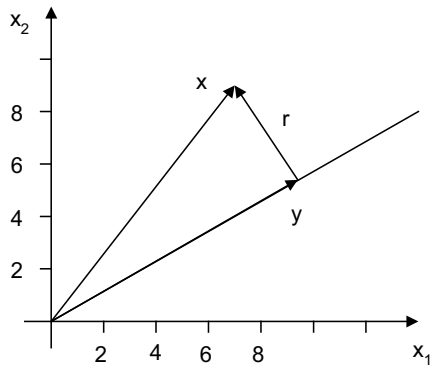

Fig. 3.6 Orthogonal Projection of a Two-dimensional Vector

The aim of PCA is to reduce dimensions in terms of orthonormal projection from this approximation fault isolation is pursued since classification comes as a goal. Moreover, for fault detection in the new sample *x(t)*, a deviation in *x* from the normal correlation would change the projections onto the subspaces (Misra et al., 2002). Consequently, the magnitude of $\hat{x}$ would increase over the values obtained with normal data.

A common technique to evaluate deviations (therefore fault presence) in this sort of approach is called the square prediction error (SPE). This is a statistic that measures the lack of fit of a model to data. The SPE statistic indicates the residual between the projection into its components retained in the model (Misra et al., 2002). This technique has been widely used in combination for fault detection in chemical processes such as presented by Patton et al., (2000).

3.5.5 Neural Network Approach

Several approaches for classification can be pursued for fault localization like clustering based techniques such as Fuzzy C Means, Fuzzy K means (Hoppner et al., 2001) or linear vector Quantification. These techniques present interesting characteristics for self-diagnose one of the most important is that related to multidimensional classification. In fact, self-organizing maps (SOM network) fulfils this characteristic.

The purpose of Kohonen self-organizing feature maps is to capture the topology and probability distribution of input data (Kohonen, 1989 and Hassoum, 1995) (Fig. 3.7). First a topology of the self-organizing map is defined as a rectangular grid (Nelles, 2001) (Fig. 3.8). Different types of grid may be used such as triangular grid, finite element grid and so on. The selected grid presents a homogenous response suitable for noise cancellation. The neighbourhood function with respect to a rectangular grid is defined based upon bi-dimensional Gaussian functions such as eqn. 3.17.

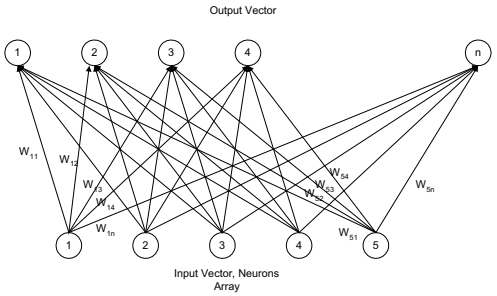

Fig. 3.7 Topology Network

$$h(i_1, i_2) = \exp\left(-0.5 * \frac{\left(i_1^{win} - i_1\right)^2 + \left(i_2^{win} - i_2\right)^2}{\sigma^2} \right) \tag{3.17}$$

where i_1 and i_2 are the index of each neuron. σ is the standard deviation from each Gaussian distribution. This distribution determines how the neurons next to the winner neuron are modified. Each neuron has a weight vector (c_i^l) that represents how this is modified by an input update.

This bi-dimensional function allows the weight matrix to be updated in a global way rather than just to update the weight vector related to the winner neuron. The use of multidimensional data characterization allows early local fault localization and its propagation as general fault presence.

Similar to other types of non-supervised neural networks such as ART2-A (Frank et al., 1998), the input vector performs an inner product with each weight vector. Having calculated every product, these are compared with each other in order to determine the largest value. This value is declared as winner. The related bi-dimensional index based upon Fig. 3.8 is calculated in order to determine how the weight matrix is modified.

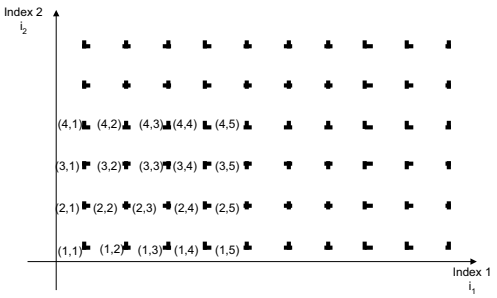

Fig. 3.8 Index Grid

The process of updating the weight matrix is based upon equation 3.18.

$$c_j^{new} = c_j^{old} + \eta * h(i_1, i_2) * \left(u - c_j^{old}\right) \tag{3.18}$$

where η represents a constant value, $h(i_1,i_2)$ is the Gaussian representation that permits the modification of neighbour neurons. Finally, u represents the current input vector.

A vigilance parameter named ρ is used in order to determine the winner from comparison between current input and every weight vector.

This whole process allows on-line classification of data based upon a defined time window by the inherent geometry of the behaviour of the system.

The use of this technique in fault diagnosis has presented several advantages as shown by Jämsä-Jounela et al., (2003). In this case fault diagnosis is performed using SOM in conjunction with heuristics rules. On the other hand, Xu et al., (2002) present a novel approach using wavelet networks and regional SOM where every sampled signal is decomposed in order to extract several features by the use of statistical analysis, thereafter, off-line feature clusters are performed for first time. Finally, on-line feature clusters is performed previous signal decomposition.

Other strategies for fault detection are based upon classical neural networks techniques such as Radial Basis Functions (RBF) and Multi-Layer Perceptron (MLP) like that presented in Fig. 3.9.

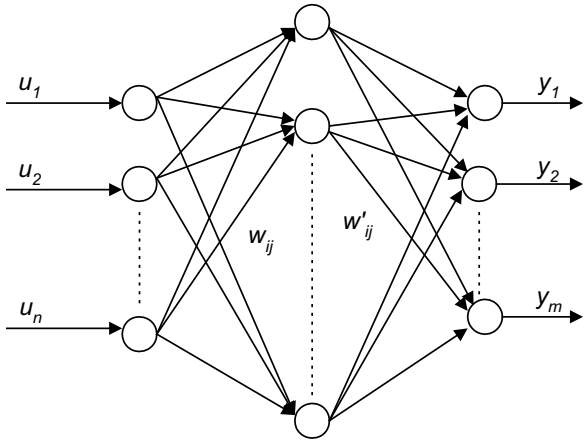

Fig 3.9 Typical RBF Configuration

For the case of RBF this is constituted following eqn.3.19.

$$y = \sum_{i=0}^{M} w_i f\left(\|u - c_i\|\right) \tag{3.19}$$

where w_i is the related weight, c_i the bias level and f functions are local one dimension guassian functions (Nelles, 2001). In this case weight updating is performed by using classical backpropagation algorithm (Werbos, 1990)

Now, for the case of multi-layer perceptron the structure of the network is defined as the following eqn. 3.20.

$$y = \sum_{i=1}^{M} w_i f\left(\sum_{j=1}^{N} w_{ij}^{'} u_j \right) \tag{3.20}$$

using the well known backpropagation algorithm as training method but considering different layers of necessary adapting weights. In this case, there are two kind of functions one for each layer representing the behaviour of the respective neurons. For last two networks learning stage (based upon backpropagation) is performed off-line giving a result of supervised neural network approach.

Having reviewed these classical neural networks strategies used for fault detection, one important issue is the referred to how to evaluate an element through this technique. Following Fig. (3.10) presents the most common configuration strategy bearing in mind that off-line stage is already been performed in order to train the network.

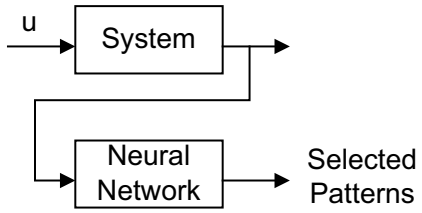

Fig. 3.10 Classical Configuration for Neural Network using for FDI purposes

Combination of different neural networks have been pursued such as Yang et al., (2004) where ART and SOM networks are combined by using the structure of first neural network and the learning strategy from second neural network. From this combination, the resultant neural network becomes suitable for unknown fault conditions. Furthermore, a review of classification accuracy based upon the modification of similarity coefficient is pursued giving a comparison methodology for this sort of techniques.

Similar strategies have been proposed like connecting these two neural networks as follows. The main approach proposed is an integration of two neural networks and a bank of unknown input observers as part of fault localization approach, which is presented in Fig. 3.1. This process performs the monitoring procedure of case study in three main stages, sampling of input and output data as well as, producing residuals based upon UIO bank and neural network supervision. The sampled information is process by a non-supervised neural network in order to be classified as pattern. The winning weight vector related to the winning pattern is classified by second non-supervised neural network.

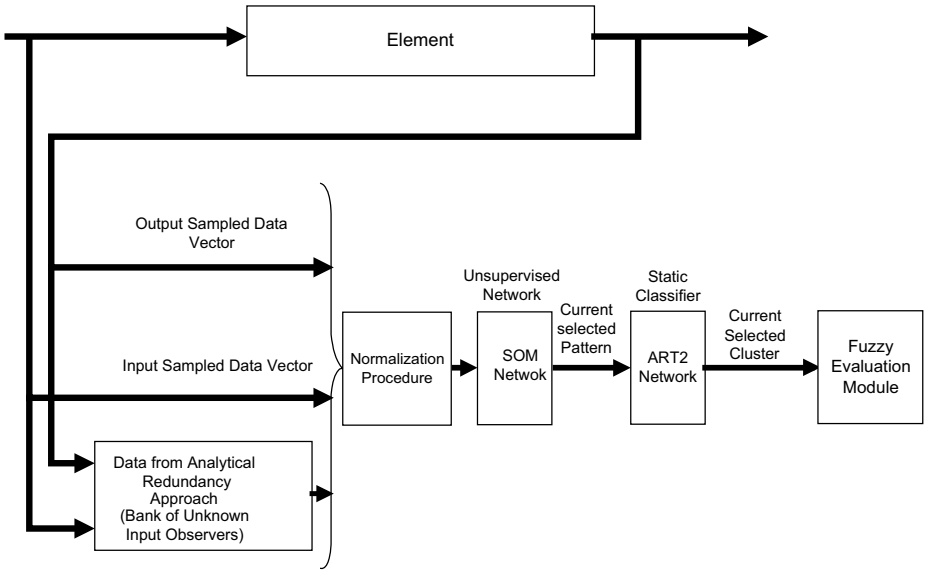

Fig. 3.11 Pursued Topology for Intelligent Fault Localization

The idea of using two consecutive neural networks is to avoid miss-classification during the presence of unknown scenarios. This goal pursues the use of a self-organizing map and adaptive resonance theory algorithms. First algorithm (Self Organizing Map) categorizes the behaviour of monitored system. The results of this categorization are evaluated by second neural network (ART2A network) in order to avoid glitches between similar categories miss-selected due to unknown scenarios.

The pursued strategy is based upon integration of an analytical redundancy approach and a fault classification technique. This fault classification approach is composed of two similar techniques in order to avoid any glitch either during transitions or during appearance of unknown scenarios. These transitions are related to several operating points from monitored element.

The data used is divided in three areas: input, output and residual data from analytical redundancy. This data is used in two stages, first, off-line in order to train both neural networks and second, on-line stage for testing this approach.

Training matrix consists of input, output and residual data and normalized between 0 and 1. In terms of scenarios, this matrix is divided in three areas as presented in Fig. 3.12. This input matrix is composed by three different kinds of variables, input, output and residuals. Each variable has *M* samples. Finally, the whole group of variables are integrated by three scenarios.

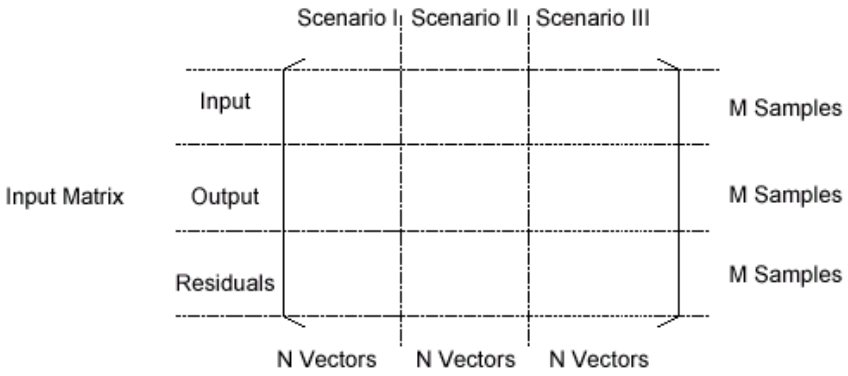

Fig. 3.12 Input Matrix Composition

During training stage each sample time window is composed of *M* samples with is directly related to a Δt time window as shown in Fig. 3.13. In this case the frequency of the fault has a bottom boundary as shown in eqn. 3.21. Where frq_{fault} represents the frequency of the monitored fault and Δt the already mentioned sampled time window.

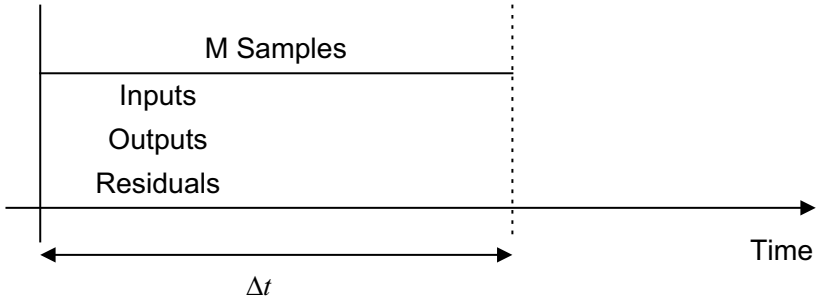

Fig. 3.13 Input matrix composed of Δt time window

$$frq_{fault} \geq \frac{4}{\Delta t} \tag{3.21}$$

It has been chosen a quarter of Δt because sampled fault information is enough to be distinguished between different patterns. Therefore the frequency of the fault can be larger than this quarter of Δt, alternatively, the top limit in terms of fault sampling is unlimited, although, the approach proposed in here would be useless to classify a fault much faster than Δt sampling window because at the time that current approach produces an output current fault can be in another stage. This top bound is still open for further research and in principle is based upon the relation between the frequency of case study and Δt time window. Formal explanation of how process monitoring is affected by sampling time window is reviewed by Campbell et al., (2004).

During on-line stage sampling time is reduced to one sample evaluated every time as depicted in Fig. 3.14.

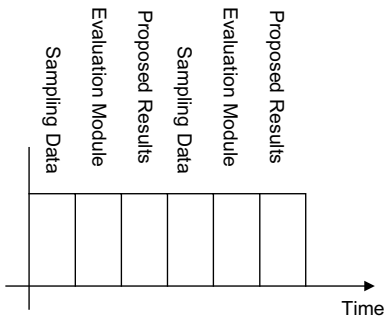

Fig. 3.14 Sampling time during on-line Performance

Having explained how sampling time window plays a key role into fault monitoring a brief description of neural networks integration is reviewed. Both neural networks are trained in cascade as shown in Fig. 3.15.

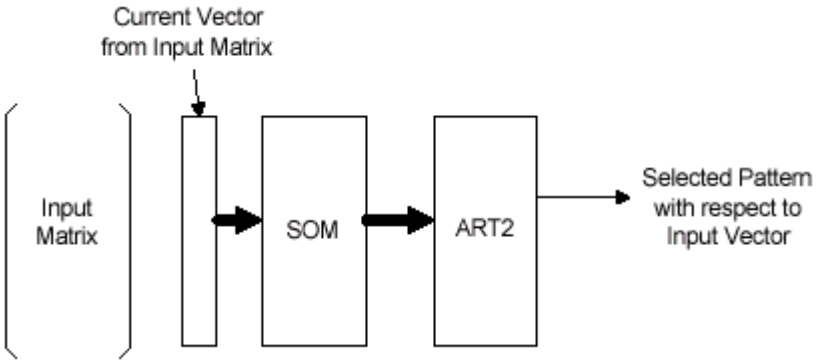

Fig. 3.15 Integration of Both Neural Networks

Both neural networks have their own weight matrix, which are initialized at 0.11 as constant number from each node. Three patterns are declared before training stage in each neural network. Second stage is related to actual on-line process information by the use of the already trained neural network.

The assumptions made on this proposal are the characteristics of the observed faults as well as the fault-free scenarios. Furthermore, it is necessary to have access to several sources of information from the monitored system. Formal knowledge of system behaviour during fault scenarios is crucial for UIO design since these are defined in terms of system response during the presence of certain unknown input. Following UIO design, input vector is conformed by three different sectors, firstly input sampled data vector, second sector is current output by sampled data vector and third sector is integrated with residual vector. Having conformed the input vector, both neural networks are trained during off-line stage. During next section a revision of each algorithm is given.

On the other hand, in order to tackle time variance classification, several solutions can be implemented. These can be time window overlapping or an increment related to the sampled input vector. Although, this last approach has as main disadvantage the scale in terms of the length of input vector and over-parameterization of represented clusters. Alternatively,

methodologies have been reviewed by Benitez-Perez et al. (2000) and Benitez-Perez et al., (1997).

Since time variant faults are the focus of this approach several strategies can be implemented such as the already explained Self Organizing Maps (Linkens et al., 1993), however, the computational cost tends to be expensive. Nevertheless, this algorithm can overcome this erratic response of case study when fault scenario is presented by a more robust pattern classification strategy based upon a global weight matrix updating procedure.

Another approach pursued in here, it is based upon two overlapped neural networks in terms of the sampled time window. Different parameters need to be established such as the time window size (t_{s*}) in terms of case study frequency response and neural network parameters (vigilance and learning values) in order to define the most suitable localization scenario. In order to produce a Fault Localization Module (named as decision-making module DMM) capable to cope with time variant systems, a group of Neural Networks is proposed. Fig. 3.16 shows this schematic approach.

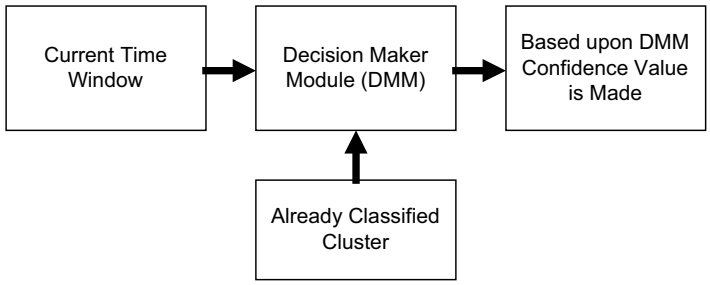

Fig. 3.16 Schematic Diagram of Decision Maker Module

Decision-making module (Fig. 3.16) defines the degree of effect from current pattern into monitored system dynamics. Thereafter, Confidence Value (CV) is produced as a percentage measure. The structure of DMM uses three ART2 networks connected as shown in Fig. 3.17.

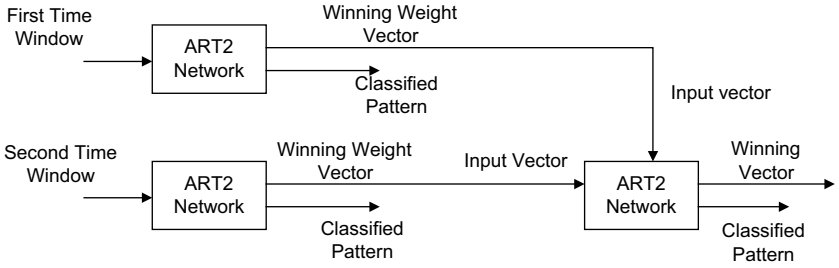

Fig. 3.17 ART2 Networks

First two networks work with two equal size consecutive time windows overlapped by fifty percent. Both networks are independent in terms of classified patterns. Third ART2 network compares the winning weights vectors from both networks in order to determine the situation of current scenario. As in a similar manner ARTMAP network performs pattern classification (Tontini G., et al., 1996). However, it differs from ARTMAP since the construction of the map performed by last element (map field) does not give any robust certainty for time variant

behaviour due to it does not conserve past information during classification of fault and fault free scenarios.

Fig. 3.18 presents how these time windows (t_{s1} first time window, t_{s2} second time window) are overlapped in order to cover time variance. It is important to define the sampling period from first and second neural networks in terms of the dynamics of case study. In fact, sampling time from first network (t_{s1}) is 50% overlapping of second sampled vector from second neural network. Fig. 3.18 shows the nominal size of each time window used to classify a scenario regardless of time behaviour of case study (Benítez-Pérez et al., 2001).

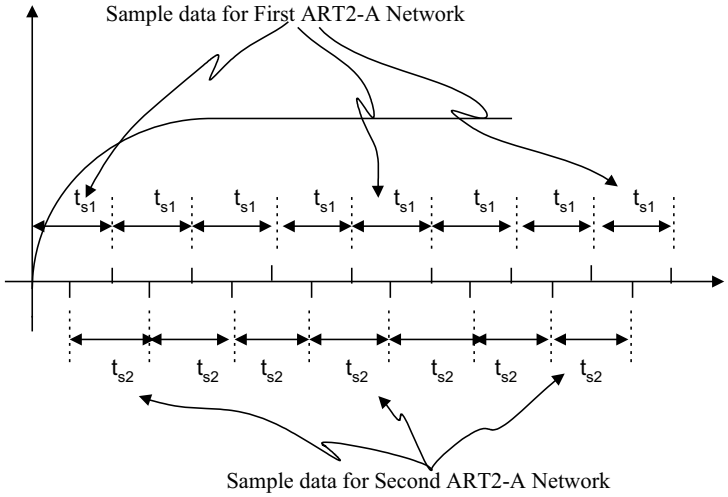

Fig. 3.18 Time Behaviour related to Dynamic Signal Response

The maximum possible sampling time value, from dynamics case study is based upon eqn 3.22.

$$t_{s*} <= \frac{t_d}{4} \tag{3.22}$$

Where t_d is defined as the inherent period from case study and t_{s*} represents either t_{s1} or t_{s2}. In this case if t_{s*} is equal to 10ms t_{s1} and t_{s2} are equal to 10ms. In order to get a good resolution in terms of fault location, t_{s*} should be smaller than or equal to a quarter of case study inherent period. This result defines the top boundary of sampling time in terms of case study. For instance, if t_{s*} is bigger than the value shown in equation 3.22, it is not possible to guarantee pattern recognition of a time variant case study. Although, sampling time is bounded from this known limit, as inherent period of case study, it does not present any restriction as bottom bound. This means that sampling period can sampled several number of case study inherent periods.

For the case of fault presence, its time response should be similar to that bounded presented with respect to fault free scenario. This means that fault scenarios with a very fast dynamics and classified as new scenarios are dependant on the resolution of t_{s*} with respect to t_d and case study fault response in terms frequency response.

Having defined how sampling period from both neural networks is pursued, it is desirable to focus on how time variant fault localization is performed. As it is known, first two neural networks classify fairly similar behaviour due to sampling time overlapping. In fact, both neural networks present similar learning values. Third neural network is the actual part of the DMM, which localizes any unknown behaviour from case study. As previously mentioned, three variables must be defined, η, ρ and t_{s*}.

The novelty of this work is based upon the overlapping time windows in order to define consistency of time variant faults. Firstly, the sampled observer event is performed as shown in Fig. 3.19.

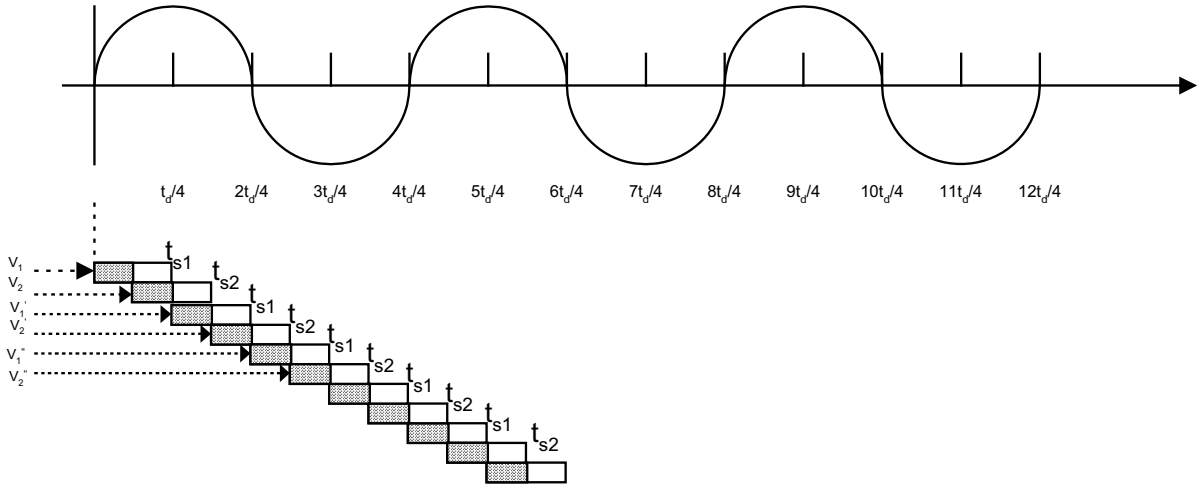

Fig. 3.19 Sampling example from current approach

Where v_1, v_2, v_1', v_2' and so on are the sampled vectors for ART1 (v_1, v_1', v_1'') and ART2 (v_2, v_2', v_2'') respectively. The shade area from both vectors is the overlapped part from the sampling procedure. Since this approximation is taken, the conformation of third weight matrix (with respect to third ART2) is presented in Fig. 3.20.

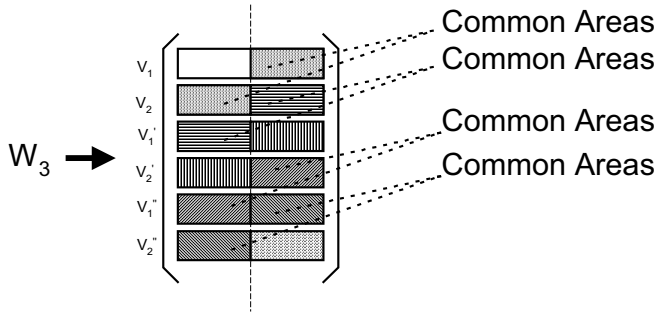

Fig. 3.20 Sampling example Related to Weight Matrix from Third ART2A Network

Where common areas are continuous vectors, therefore for fault free scenarios the selection of two similar weight vectors is expected. Now, when a fault appears this approach behaves in terms of sampling structure as a filter that it would take 1 ½ sampling cycles to declare the presence of an abnormal behaviour even in conditions of early stages (Fig. 3.21).

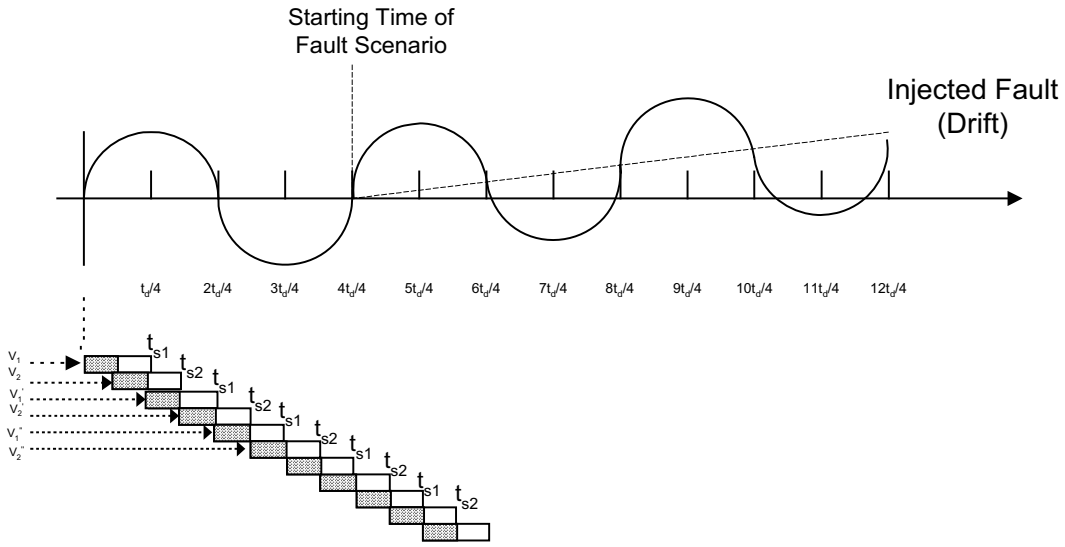

Fig. 3.21 Current Sampling Approach during Fault Scenario

In this case, fault presence is first captured for ART1, then ART2 detects the presence of the fault. Thereafter, third ART network processes the selected weight vectors where the on-line classification is performed due to new information presence. For this case presented in Fig. 3.21, one winning vector will arise because fault is monotonically increased.

Classified patterns from third network are the representative values of current scenario either fault or fault free with the only restriction of top bound sampling period. Third neural network, firstly classifies the winning vector from first neural network, afterwards, it classifies second neural network winning vector. This last classification is the representative of current scenario. If there is a fault, this selected pattern represents the fault.

Third ART2 network has the advantage of producing a weight matrix with the most representative patterns of certain scenario. The related winning vector is processed by a Mamdami fuzzy logic system to generate a confidence value (CV).

The final inference machine that produces CV is presented in Table 3.2. This machine has been built under the heuristic knowledge of the designer. The number of components is constant; each component has been normalized between 0 and 1. The number of components is a direct representation of the number of elements sampled by first two neural networks.

Component 1	Component 2	Component 3	Component 4	Result
High	High	High	Low	100%
Med	Med	Med	Low	80%
Low	High	High	Low	50%
Low	Med	Med	High	40%
Low	Low	Low	High	10%

Table 3.2 Fuzzy Logic Table related to Confidence Value

The universe of discourse for each component has been divided by three similar boundaries named high, middle and low. Final result is related to the correspondent value of the already known behaviour of the monitored system.

Different patterns have been defined with respect to a nominal value (Fig. 3.22). During on-line performance if a new pattern appears this is declared as a 0% because it represents an abnormal situation that it has not been defined previously giving a safe response for this new scenario. This approach has the capability of classification of known and unknown scenarios with just a top sampling boundary. However, fuzzy logic approach requires further work in order to overcome an oscillated response due to current injected fault.

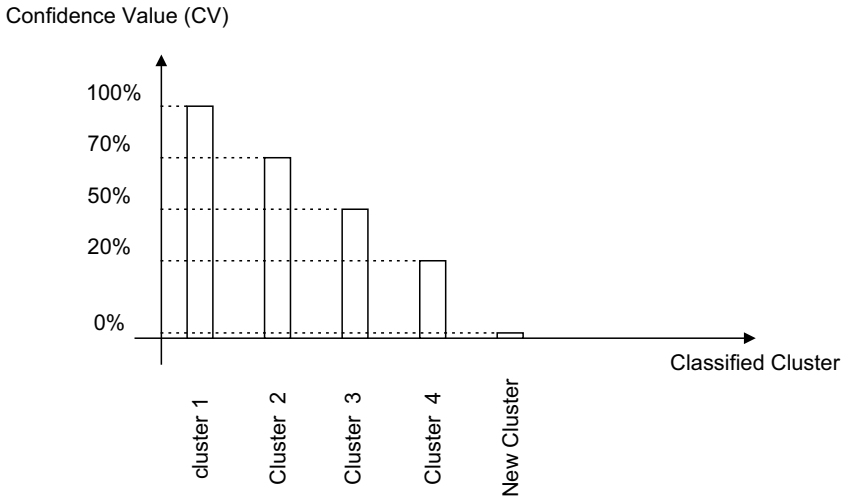

Fig. 3.22 Relation between Patterns and CV

3.5.6 Logic as Fault Diagnosis Strategy

Other techniques based upon logic such graphs techniques, predicate logic techniques and fuzzy logic based techniques are suitable for fault diagnosis since can separate the behaviour of current system by classifying real time system response through residual evaluation like structural analysis where residuals combination give a particular fault signature as presented by Frank et al., (2000), giving faults several characteristics depending on residual responses. Other kind of logic is fuzzy logic where residuals again are evaluated through fuzzy system in order to declare certain fault behaviour, classical application is shown in Fig. 3.23.

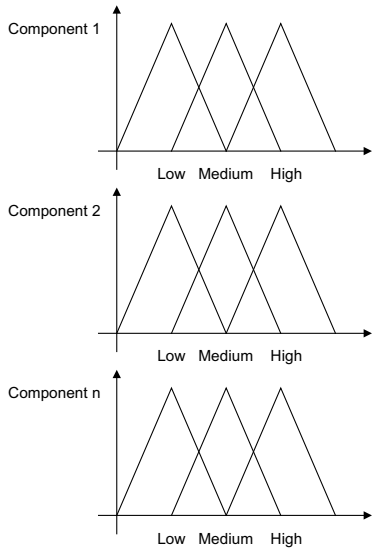

Fig. 3.23 Classical Fuzzy Logic Structure

Where different components play several roles like characterization of signals into membership functions which are labeled in this particular case as *Low, Medium* and *High*. In this case, the level of approximation is quite related to the use of these membership functions in order to define the behaviour of the observed signals. Moreover the use of the inference machine related to fuzzy logic gives the representation from the response of this kind of technique.

Similar strategies like structural analysis present the advantage of fault evaluation in order to characterize system behaviour under fault conditions a good review of this strategy is presented in Blanke et al., (2003).

3.5.7 Heuristic Confidence Value Definition

Having defined the use of different neural networks as an approach to classify unknown scenarios, it is important to introduce a heuristic measure as result of this evaluation. This heuristic measure defines how a current scenario has been degraded. This measure is based upon a fuzzy logic module that evaluates the winning weight vector related to the classified pattern in second neural network.

The use of fuzzy logic presents the most suitable mechanism in order to evaluate those already classified patterns. Although, information used to define fuzzy module should be rich enough to avoid any non-desirable response. To reproduce this information into fuzzy knowledge, it is necessary to follow the typical procedure of normalization, fuzzification, inference machine performance and defuzzification. To incorporate further knowledge it is necessary to use off-line learning techniques. Different techniques are available such as clustering or genetic algorithms (Mitra, 1994).

The use of Mamdami based approach (Driankov, et al., 1998) is pursued due to knowledge representation and low computational cost. This module produces a percentage measure that represents the response of the peripheral element with respect to current scenario (either fault free or fault scenario), which is named as Confidence Value.

This measure classifies the behaviour of peripheral element under the presence of a fault. Confidence Value shows the degradation of the element with respect to the output, input and parameters. This module performs the evaluation of the selected pattern in order to produce a percentage representation of current behaviour. As mentioned before, the procedure that fuzzy logic acquires the knowledge it is the key issue. Different methodologies can be followed. For instance, the use of heuristic knowledge is the must straight forward approach. Alternative strategies such as genetic algorithms or mountain-clustering are suitable in order to define the most accurate knowledge for specific scenarios. In here the followed approach is based upon heuristic knowledge. Confidence value has a continuous range from zero (catastrophic situation) to one (fault-free scenario). Its graphical representation is depicted in Fig. 3.22.

The inference machine that produces CV is presented in Table 3.2. This machine has been built under heuristic knowledge of the designer. Ten patterns have been defined with respect to a nominal value from input vector. During on-line performance if a new pattern appears this is declared as a 0% because it represents an abnormal situation that it has not been defined previously. The number of components is constant; each component has been normalized between 0 and 1. This range has been divided by three similar boundaries named high, middle and low using triangular membership functions overlapped by fifty percent (Fig. 3.24). Final result is related to the correspondent value of the already known behaviour of the monitored system.

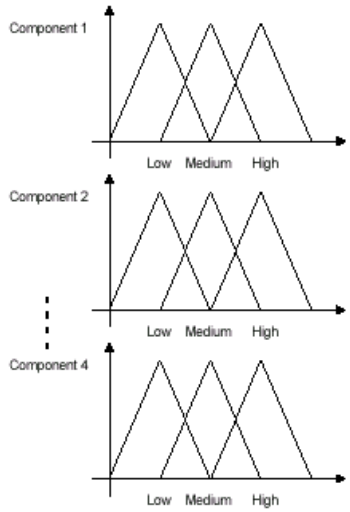

Fig. 3.24 Fuzzy Membership relation

This module is connected in cascade mode to neural network, the input of this fuzzy evaluation module is referred to each component of winning weight vector from current

selected pattern. This means that each component of winning weight vector has an unvalued relation to each component from the fuzzy module (Fig. 3.25).

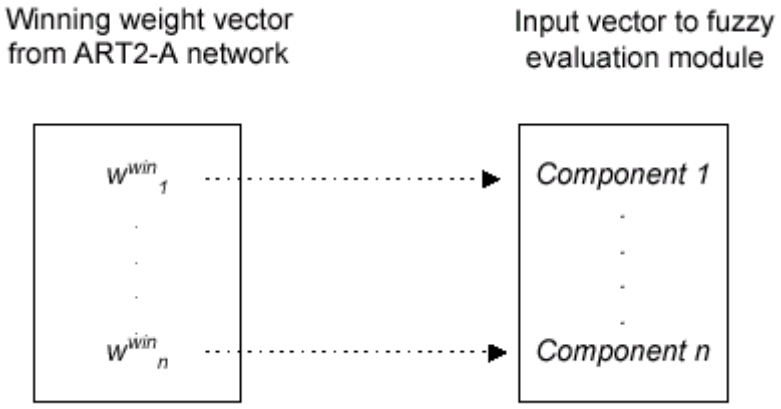

Fig. 3.25 Relation between weight vector and fuzzy module

3.6 Conclusions

This chapter has presented an overview of several techniques for fault diagnosis in order to enhance autonomy amongst peripheral elements like actuators and sensors. It has been shown that unlike one particular well defined and mature strategy the combination of various issues strength the capabilities for fault detection as shown at the combination of two neural networks.

Performance measures are necessary to develop in order to determine weather any strategy is feasible for fault diagnosis. One particular strategy based on structural analysis permits to determine fault appearance through elements monitoring according to related constrains inherent on system perform. Since this particular information is available through either structural analysis or confidence value at last both techniques will only report those observed faults where non-measurable faults (hidden faults) are still an open research field.