Fault Tolerance in Large Scale Systems: hybrid and distributed approaches

Ph.D. Thesis

Dr. Marta Capiluppi

Supervisor: *Prof. Claudio Bonivento*
Co-Supervisor: *Prof. Manfred Morari*
Coordinator: *Prof. Claudio Melchiorri*

XIX Ciclo
ING-INF/04
A.A. 2003 – 2006
University of Bologna

Keywords:

fault tolerance, fault diagnosis, distributed systems, structural analysis, hybrid systems.

Contents

List of Tables

List of Figures

Introduction

Physical systems can fail. The main aim of fault tolerance is to cope with this situation. When a system fails, it is not able to achieve its goal anymore, because the function for which the system is designed is not produced with the required performances. The cause of the failure is generically called fault. Faults can occur in different parts of the system and affect it with different levels of severity. A system is fault tolerant when despite the occurrence of a fault it is still able to perform its function. Hence fault tolerance provides the tools to detect the fault, to locate it and estimate it and to decide a policy for react to the fault.

The fault tolerant problem can be divided in two tasks: fault detection and isolation (FDI) and control redesign. FDI produces a diagnostic result including detection and location of the fault, and if possible an estimate of the dimensions of the fault. The first attempts to reach this objective were based on hardware redundancy: the outputs of identical components are compared to check if one of them failed. An evolution of this method is analytical redundancy, which is based on the same concept, but with the mathematical model of the system. In analytical redundancy the constraints among the system variables are compared to check inconsistencies. This was also called model-based FDI at the beginning, because the mathematical model of the system was the basis for fault detection. Nowadays the systems are so complex that their model is not always known. This leads to new approaches for FDI, based on structural and functional considerations, i.e. on a qualitative analysis of the system. These methods are borrowed from the qualitative reasoning in artificial intelligence, with the aim of understanding the human common sense to catch the interactions between physical phenomena, without knowing the quantitative aspects. Qualitative models are hence abstractions of the system behavior, they represent qualitative relations among physical systems.

The complex modern systems are usually distributed, i.e. they are composed by subsystems connected through a network. These subsystems achieve different functions and communicate to reach a common goal. The subsystems composing a distributed systems can be computers or dynamical systems. Depending on their nature and on the aim of their analysis, they can be represented by discrete event or hybrid systems. A discrete event representation is more used for a higher level description, when only the discrete dynamics and the communication policy is considered. The hybrid representation involves also the continuous dynamics of each subsystem and the interaction between the continuous and the discrete dynamics. Indeed discrete event systems are discrete-state, event-driven systems, while hybrid systems are systems in which continuous and discrete dynamics interact.

Fault tolerance in these large scale systems is quite complex, because it has to consider problems of propagation of the fault effect and problems of scaling. Moreover to achieve fault tolerance without increasing the computational burden some decentralized methods have to be used. These methods can cope with the distributed nature of the systems, with the final aim to decentralize the tasks of fault diagnosis and control redesign.

The aim of this thesis is to introduce new methods for dealing with fault tolerance in complex,

distributed systems. In Chapters 1 and 2 some background tools are recalled. In Chapter 1 the notions of fault tolerance, with some methods for fault diagnosis and control redesign are reviewed. In Chapter 2 large scale systems are introduced, with a deeper overview on distributed, discrete event and hybrid systems. In Chapter 3 an architecture for fault tolerant control of distributed systems in introduced. In Chapter 4 some considerations and issues for fault detection in hybrid systems are presented.

Chapter 1

Fault Tolerance and Fault Tolerant Systems

A *fault* is a deviation of the system structure or the system parameters from the nominal situation [5]. This implies that after the occurrence of a fault the system will have a behaviour which is different from the nominal one. Hence *Fault Tolerance* is the property of reacting to faults. In particular the analysis of fault tolerance consists in establishing if a given system is still able to achieve its tasks after the occurrence of a given fault, whereas the synthesis of fault tolerance resides in providing a given system the tools to react to a given faulty situation. Therefore the knowledge of the set of faults that can occur in the system and of the objectives the system has to achieve are required for fault tolerance. For this reason in section 1.1 the different possible classifications of faults are introduced, presenting some techniques to obtain the kinds of fault that can occur in a system and a way of modelling them.

A system is *Fault Tolerant* (FT) if after a fault it is able to recover its original task, with the same or degraded performances. The fault tolerance problem is generally divided into two steps:

Fault Diagnosis: it consists in the detection of the occurred fault and in its isolation (identification).

Control redesign: the diagnostic information is used to adapt the controller to the faulty situation in order to achieve again the objective of the system.

These two steps will be better explained in sections 1.2 and 1.3 of this chapter. A general architecture for fault tolerance is represented in fig. 1.1 where it is shown how fault tolerant control extends the feedback loop: the output of the system is used by the diagnoser together with the input to decide if a fault has occurred, then the diagnostic information is transmitted to the controller redesign block that changes the control policy to face the fault.

Fault Tolerance can be considered as a measure of *Dependability* of a system, where dependability is defined as "the index of the quality of service a system delivers to its users during an extended interval of time" (see [43]). Other measures of dependability can be stated:

Safety describes the real absence of danger, meaning that if a critical fault occurs the system has to shut down not to provoke dangerous conditions. When this property is satisfied the system is called *fail-safe*.

Reliability is the probability that a system will perform the function for which it is designed for a given period of time in nominal conditions. In this context, fault tolerance can help improving the overall reliability of the system.

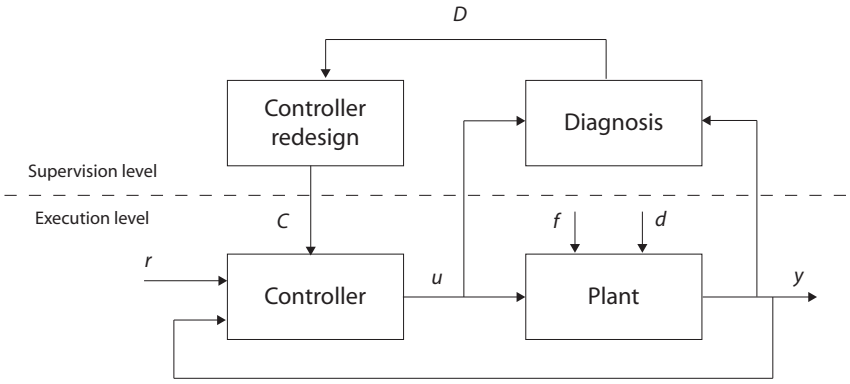


Figure 1.1: Fault tolerant control architecture: r is a reference signal, u is the control signal, y is the output signal, f is a fault signal, d is a disturbance, D is the diagnostic information and C is the control redesign information.

Availability is a measure of the correct service delivery regarding the alternation of correct and incorrect services.

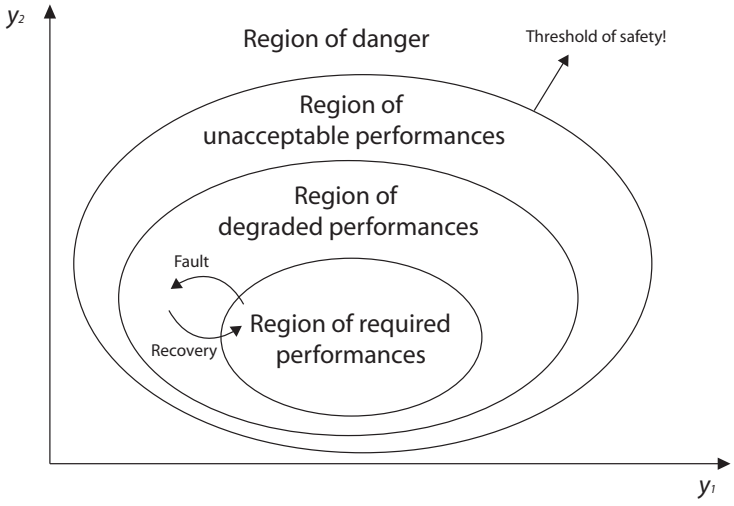


Figure 1.2: Fault tolerance vs Safety.

Hence finally a dependable system is a fail-safe system with high availability and reliability. But what is the real difference between safety and fault tolerance? In fig. 1.2 the concept is clarified using performances regions. If the system performances can be quantified by the two variables y_1 and y_2, then it is possible to define a region where the performances are the required ones, i.e. where the system remains during the nominal operation behaviour. The occurrence of a fault can bring (and usually does) the system in the region of degraded performances. When this happens the fault tolerant controller should be able to recover the system to bring it again the region of required performances, or if not possible at least to keep it in the region of degraded performances, without exceeding this threshold. Outside the region of degraded performances there is the region of unacceptable performances where the system should never move, and the task of the fault tolerant controller is exactly to prevent it. Outside this region there is

the region of danger. The purpose of a safety system is to interrupt the operations when this region is reached. This means the fault tolerance and safety work in two different regions of the performance space and they are called into the play when two different threshold are crossed. Therefore in industrial plants they are often threaten separately. This separation is used to design fault tolerant controllers without the need to meet safety standards.

Fault tolerance and fault tolerant systems have been studied since the late '70s, as in [37] where fault detection for chemical processes is introduced, and later in [58]. One of the first surveys on fault detection is [39], which is dated 1984, and where some methods based on modelling and estimation are introduced. Much later the interesting book [57] collects some results on Fault Detection and Isolation (FDI) methods. Control redesign has been studied later than fault detection, due to the technological improvements. A survey on the late '90s situation is given in [56]. Thereafter many new techniques in this field have been presented, but it is almost impossible to write a complete bibliography on them. Nevertheless throughout this work some of the most interesting and recent methods will be cited and used. For a complete outline of the recent improvements in this field, it is worth citing some works which will not be used in the following. In [60] a quite new approach to fault detection in industrial (batch) systems is introduced. In [42] an overview on fault tolerant techniques for flight control is presented. A new approach for fault accommodation in nonlinear system is introduced in [40].

Most of the notions recalled in this chapter are based on the book [5], where a survey of the most recent contributions to the fault tolerance problem are reported, and on [66] for fault tolerant systems.

1.1 Faults Classification and Modelling

From the definition at the beginning of this chapter, a fault changes the structure of the system or its parameters. Structural faults generally change the set of component interacting in the system or the relation between the plant and the controller. Parametrical faults, instead, act on the input/output properties of the plant, causing a deviation from their nominal conditions. This situation leads to a change in the performances of the closed-loop system with consequence on the achievement of the system function.

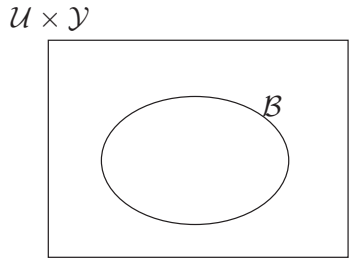


Figure 1.3: System behaviour.

Practically it is possible to recognize that a fault has occurred when given a known input the output of the system is different from the one expected. In fig. 1.3 the input/output (I/O) space of the system is represented, where $\mathcal{U}$ is the space of input signals and $\mathcal{Y}$ is the space of output signals. The behaviour of the system is represented by the set $\mathcal{B}$ of all the admissible input/output pairs (u, y). For dynamical systems in discrete time these pairs are represented by temporal sequences $U = (u(0), u(1), \ldots, u(k_h))$ and $Y = (y(0), y(1), \ldots, y(k_h))$, where k_h

represents the time horizon in which the sequence is considered, and often the actual instant of time. In this case the behaviour of the system has a temporal connotation.

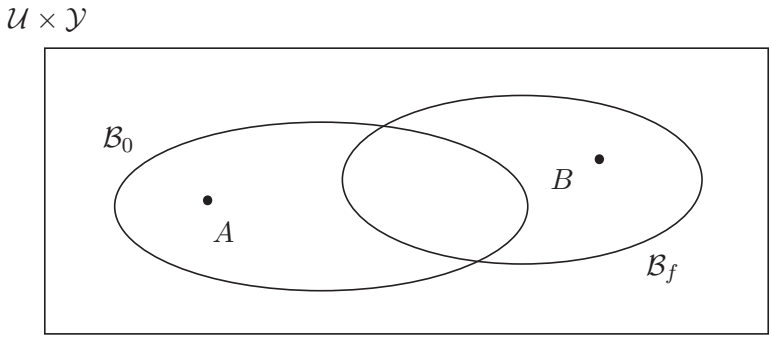


Figure 1.4: Behaviour of a system subject to faults.

In fig. 1.4 the effect of the fault on the system behaviour is represented: in the I/O pair space $\mathcal{B}_0$ represents the nominal behaviour, $\mathcal{B}_f$ represents the faulty behaviour (for fault f). When a given input U is applied to the system, if the system is in nominal conditions it produces output Y_0, if it is faulty it produces output Y_f. In fig. 1.4 the point $A = (U, Y_0)$ represents the nominal I/O pair, the point $B = (U, Y_f)$ represents the faulty I/O pair. To detect and isolate the fault, the related I/O pair has to be in $\mathcal{B}_f$ like point B but not in the intersection of $\mathcal{B}_0$ and $\mathcal{B}_f$.

Different classifications of faults exist. The more used is based on the part of the system they affect:

- **Plant Faults:** they change the dynamical I/O properties of the system;
- **Sensor Faults:** they cause errors on the sensors outputs, but not on the plant dynamics;
- **Actuator Faults:** they corrupt the actuating signal, i.e. they modify the influence of the controller on the system.

The quantitative classification divides the faults in **complete** (total loss of the component) and **partial** (partial loss of the component). Following a qualitative classification the faults can be **abrupt** (sudden breakdown) or **incipient** (drift). Note that abrupt faults usually have an important role in the so called safety-relevant systems, because they have to be detected before the system becomes safety-critical, i.e. before their effect on the system cannot be recovered with fault tolerance anymore. Incipient faults, instead, are more related to maintenance problems, where the important task is to avoid drift effects, thus fault detection time is not so important, but these faults are usually small and more difficult to detect.

From a modelling point of view, the faults are classified in **additive**, if they are considered as additional external signals, i.e. unknown inputs, and **multiplicative**, if they are seen as a parameter deviation.

In general continuous-variable continuous-time models the faults are represented in the state-space form:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + E_x d(t) + F_x f(t), x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + E_y d(t) + F_y f(t) \end{aligned}$$

for additive faults f, where d is a disturbance. Multiplicative faults can be represented like:

$$\begin{aligned} \dot{x}(t) &= A(\theta)x(t) + B(\theta)u(t), x(0) = x_0 \\ y(t) &= C(\theta)x(t) + D(\theta)u(t) \end{aligned}$$

where θ is a parameter vector with nominal value θ_0 and different values representing parametric faults.

Depending on the model of the system many other different ways of introducing faults in the representation are possible. For a complete survey refer to [5].

Finally for the sake of clarity, it is necessary to explain what is the difference among fault, disturbance, uncertainty and failure. Disturbances and uncertainties are usually represented similarly to the faults, in additive and multiplicative form. Nevertheless these nuisances are considered in the controller design phase, using filters and robust methods. Faults are more severe changes that cannot be fixed by the same controller of the nominal behaviour, but they need to be detected and their effect has to be handled with remedial actions involving some changes to the control law. A failure can be considered as the effect of the fault, because it describes the inability of a system or a component to accomplish its function (task). In this sense a failure is more severe of a fault and must be avoided, using fault tolerance to prevent the fault before it causes a failure. In chapter 3 this notion of failure will be extended.

1.2 Fault Diagnosis

As sketched in fig. 1.1 a fault tolerant system has to detect the occurrence of a fault to react with the appropriate measure. Moreover the fault needs to be located and the kind of fault identified. These three tasks are accomplished by the diagnoser:

Fault Detection: it aims at deciding if a fault has occurred, i.e. it detects the instant of time in which the system becomes faulty.

Fault Isolation: it finds out in which part of the system the fault has occurred, i.e. the location of the fault is determined.

Fault Identification and Estimation: the kind of fault and its magnitude are determined, the fault severity is stated.

The diagnostic task is usually performed online, meaning real-time, using the measurement data available from the process. Hence the diagnostic problem can be defined as:

Definition 1.1. Diagnostic Problem.
For a given I/O pair (U, Y), find the fault f.

The basic idea of a diagnostic algorithm is to compare the actual behaviour of the system with the model of the nominal one. This is also called consistency-based diagnosis. The element to perform the diagnostic task are then:

- the measurement information (U, Y) up to the actual instant of time;
- the nominal behaviour $\mathcal{B}$ of the plant;
- the faulty behaviour(s).

As shown in fig. 1.5 the point A represents the I/O pair of the nominal system, because it is inside the nominal behaviour set $\mathcal{B}$, while the point C represent the system behaviour after fault f. Since C is out of the nominal behaviour, the fault f is detectable. Fault isolation and identification requires also the knowledge of the sets representing the system behaviour after the fault, i.e. the possible I/O pairs that can be measured after a fault occurrence. Fig. 1.6 represents

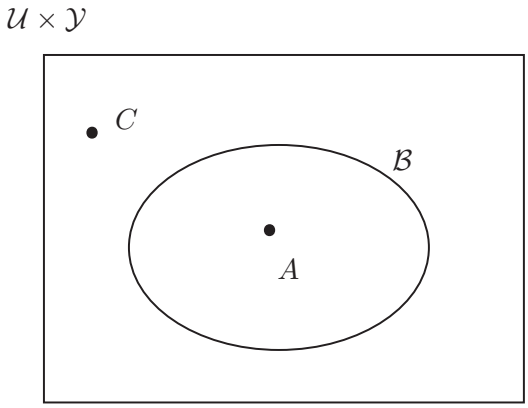


Figure 1.5: Fault Detection and system behaviour.

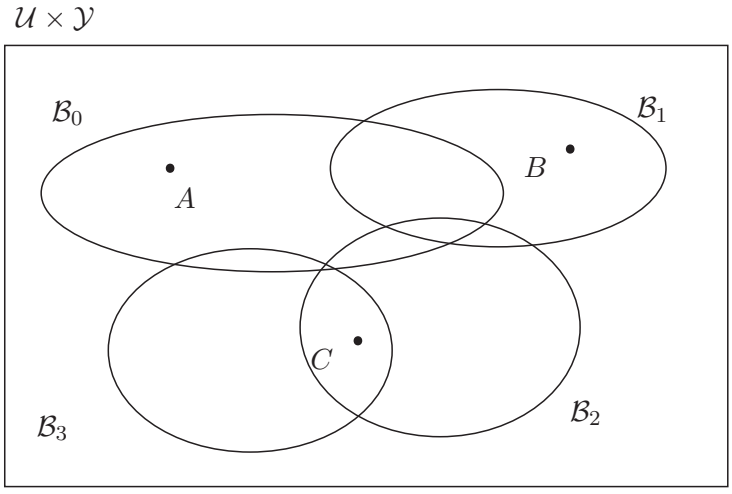


Figure 1.6: Fault Detection and Isolation and Diagnosability.

the I/O space of a system in which faults f_1, f_2 and f_3 can occur, with the associated behaviour sets $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ respectively ($\mathcal{B}_0$ represents the nominal behaviour). If the measured I/O pair is represented by point A the system is in nominal conditions, if it is represented by point B fault f_1 is detected and identified, if it represented by point C a fault can be detected but it is not possible to distinguish between fault f_2 and fault f_3. This problem is called **diagnosability** and it is a property of a given system subject to given faults. Diagnosability has different definitions depending on the kind of system under consideration, thus it is impossible to give a general definition for it (see [5]). Nevertheless to understand the concept a definition can be given for continuous-variable continuous-time deterministic systems:

Definition 1.2. Diagnosability
A faukt f is said to be strongly diagnosable if there exists a stable residual generator such that the residual signal $r(t)$ reaches a non-zero steady-state value for a fault signal that has a bounded final value different from zero.

Being consistency-based diagnosis a general concept, it can be applied to any kind of systems, using different strategies. For continuous-variable systems the generation of **residual signals** is used. As explained in fig. (1.10 pag 17) the consistency of the real-time system with its model can be checked at each instant of time using the residual signal

$$r(t) = y(t) - \hat{y}(t).$$

In faultless case the residual signal vanishes, or more realistically it is very close to zero, due to noise. This problem is usually solved establishing a threshold that the residual should cross to say that a fault has occurred. For this reason the possible faults for a certain system need to be studied giving also their possible magnitudes and their effects on the system.

Generally even the diagnostic task can be split in two subtasks:

Residual Generation: Residuals are generated using the I/O pairs and the model of the system. This determines the degree of consistency between the actual and the known model of the plant.

Residual Evaluation: The residual is evaluated in order to detect, isolate and identify the fault.

Many different residuals, sensitive to different faults, can be designed and evaluated, singularly or jointly, to detect and isolate the faults. The consistency method is based on the so called *analytical redundancy*, i.e. it is based on the model of the system. Hence it avoids the use of *physical redundancy*, because there is no need of adding new components to check if the result is the same for each of them. Analytical redundancy allows a cheaper fault diagnosis and its importance is growing up in actual industrial plants.

The above explained procedure is also called **model-based FDI** and it is represented in fig. 1.7.

The first step of the detection procedure is the residual generation and it is basically performed in two ways: using state estimation techniques (parity space, dedicated observer and detection filter approaches) or using parameter identification methods. The second step is performed using decision functions, such as norms and likelihood functions. In the following a flavour of the different methods for residual generation is given, based on the work in [30]. For the sake of brevity robustness issues and residual evaluation methods are omitted, a good survey can be found in [5].

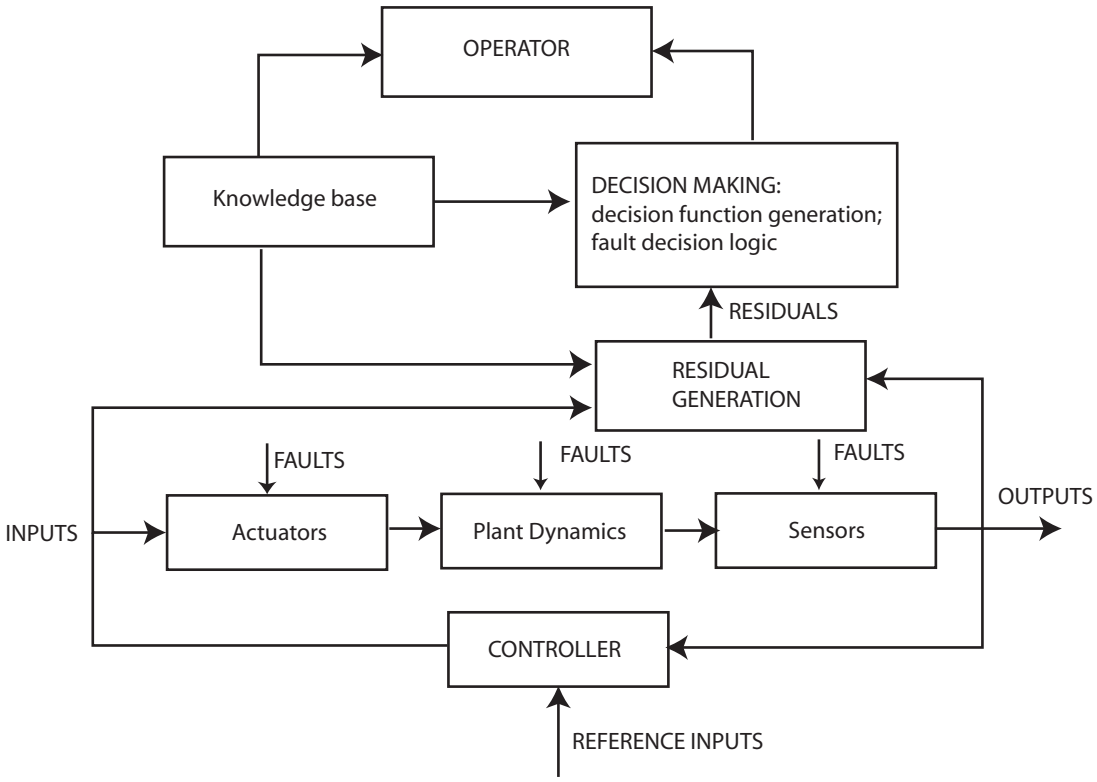


Figure 1.7: Model-Based FDI.

For all the presented approaches consider a continuous system of the kind:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + Ed(t) + Kf(t) \\ y(t) &= Cx(t) + Fd(t) + Gf(t) \end{aligned} \tag{1.1}$$

where d models the unknown inputs, Kf the actuator and component faults and Gf the sensor faults.

1.2.1 Parity Space Approach

This method uses the analytical redundancy relations, exploiting the parity (i.e. the consistency) of the mathematical model. This is checked using the system measurements. A fault is declared to be occurred if some bounds b_i are crossed.

Two kinds of redundancy can be used:

- **direct redundancy**: based on algebraic relations among redundant sensor outputs;
- **temporal redundancy**: based on differential or difference relations between sensor outputs and actuator inputs.

If direct redundancy is exploited, the parity space method can be outlined as follows. Let the algebraic measurement equation be:

$$y_m = My + \Delta y$$

where y_m is the measurement vector, y the true measurement value and Δy the error vector, i.e. $\Delta y_i > b_i$ defines a faulty operation indicated by the i-th measured variable.

To detect Δy the vector y_m can be combined to a set of linearly independent parity equations:

$$p = Vy_m$$

with V such that

$$VC = 0$$

$$V^TV = I - C(C^TC)^{-1}C^T$$

$$VV^T = I.$$

Hence

$$p = V\Delta y.$$

This means that the parity equation contains only the errors due to the faults. The columns of V define distinct fault directions, associated with each measurement: the i-th column of V gives the direction along which p lies if $\Delta y = \Delta y_i = [0 \cdots 0 \Delta y_i 0 \cdots 0]^T$. The residual vector can be written as

$$r = y_m - M\hat{y}$$

where

$$\hat{y} = (M^TM)^{-1}M^Ty_m$$

is the least squares estimate of y. The residual is related to the parity vector p by the equation:

$$r = V^Tp.$$

Finally, given q redundant measurements of a process variable y and symmetric error bounds $b_1, \ldots, b_q$ characterizing the faulty behaviour, the FDI problem consists in finding an estimate $\hat{y}$ of the process variable from a most consistent set of measurements and identify the faulty measurements by parity checks. Note that to detect a fault among p components at least $p-1$ parity relations are needed.

Now the case of temporal redundancy relations is sketched. Consider the system

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{1.2}$$

The parity space of order s is defined as

$$P = \left\{ v | v^T \begin{bmatrix} C \\ CA \\ CA^s \end{bmatrix} = 0 \right\}.$$

For parity checks every vector v can be used at any time k:

$$r(k) = v^T \left[\begin{bmatrix} y(k-s) \\ \vdots \\ y(k) \end{bmatrix} - H \begin{bmatrix} u(k-s) \\ \vdots \\ u(k) \end{bmatrix} \right]$$

with

$$H = \begin{bmatrix} 0 & & & & & 0 \\ CB & 0 & & & & 0 \\ CAB & CB & 0 & & & 0 \\ \vdots & & & & & \\ CA^{s-1}B & \cdots & \cdots & CAB & CB & 0 \end{bmatrix}$$

Substituting the state equations of (1.2) in $r(k)$:

$$r(k) = v^T \begin{bmatrix} C \\ CA \\ CA^s \end{bmatrix} x(k-s).$$

This means that instead of checking the consistency of the whole mathematical model, only individual relations that are a part of the model are checked.

1.2.2 Dedicated Observer Approach

This method is based on the state estimation, through banks of observers or Kalman filters. The estimation error or the innovation are used as a residual for the detection and isolation of the faults.

To better explain this concept, consider the system represented by (1.1). An observer for this system is given by

$$\begin{aligned} \dot{\hat{x}} &= (A - HC)\hat{x} + Bu + Hy \\ \hat{y} &= C\hat{x} \end{aligned}$$

where H is the feedback gain matrix chosen to achieve certain performances of the observer. Defining $\varepsilon = x - \hat{x}$ the state estimation error and $e = y - \hat{y}$ the output estimation error we obtain:

$$\begin{aligned} \varepsilon &= (A - HC)\varepsilon + Ed + Kf - HFd - HGf \\ e &= C\varepsilon + Fd + Gf \end{aligned}$$

Since e is a function of f and d but not of u, it is a good candidate for being used as a residual.

The **fault detection filter** method is a full order state estimator where H is chosen in a particular form.

The use of estimation schemes and Kalman filtering allows the fault detection of a variety of systems, due to the many different methods of estimation present in literature. Some of them are able to cope with nonlinearities, it is possible to use banks of estimators to improve the effectiveness of the fault detection task, there is also the possibility of introducing some stochastic elements. The interested reader is referred to the wide literature and to the above cited works.

1.2.3 Parameter Identification Approach

This method has been introduced by [39] as an alternative way to the state estimation methods. The basic idea is to estimate the parameters of the mathematical model following these steps:

1. The system is represented through an input/output model
$$a_n y^{(n)}(t) + \ldots + a_1 \dot{y}(t) + y = b_0 u(t) + \ldots + b_m u^{(m)}(t).$$
2. The relation between the model parameters θ_i and the physical parameters p_j is determined:
$$\theta = f(p).$$
3. The model parameter vector θ is identified using the input and output of the actual system (or of one of its components).
4. The physical parameter vector $p = f^{-1}(\theta)$ is obtained.
5. The vector of deviations Δp is computed.
6. The detection of the faults is performed exploiting the relations between faults and changes in the physical parameters Δp.

This method is particularly useful in case of incipient faults, because it is able to detect small changes in the parameters.

1.3 Fault Tolerant Control

The second step to achieve fault tolerance in a system is controller redesign, or fault tolerant control.

Fault tolerant control is just one of the fault tolerant problems, which refers to the achievement of the control specifications even if a fault occurs. Other fault tolerant problems can be stated if other system properties are considered, e.g. if observability is the considered property, fault tolerant estimation is required. In this section only fault tolerant control is described, but fault tolerant estimation is based on the same principles and obvious extensions can be figured out. A fault tolerant estimation problem will be presented in chapter 3.

The basic idea behind fault tolerant control is the implementation of a control law able to achieve system objectives in spite of the occurrence of a fault (see [66]). Then the fault tolerant control problem is above all a control problem. This consideration leads to the need of defining the standard control problem.

The control problem aims at finding a control law in a set U such that:

1. the controlled system is able to achieve the control objectives O,

2. the system behaviour satisfies a set of constraints C.

The solution of the control problem is defined by the triple $< O, C, U >$. If uncertainties are considered, the parameter vector θ is introduced, from which the constraints depend, hence the control problem becomes $< O, C(\theta), U >$.

As above explained the faults can have different effects on the system. These effects are now cast in the fault tolerant control problem. The occurrence of a fault generally does not change the system objectives, because the main idea of fault tolerant control is to try to reach them even in presence of a fault. When this is not possible, i.e. when the system is not fault tolerant, it is necessary to adopt the policy of objective reconfiguration. On the contrary a fault changes the constraints of the control problem, because it acts either on the parameter of the system or on the structure of the constraints themselves. The nominal control problem will then be $< O, C_n(\theta_n), U >$ and the faulty one $< O, C_f(\theta_f), U >$. The fault occurrence can also change the set of admissible control laws (i.e. if the fault occurs in computing and communication devices): after a fault the set of control laws will then be U_f (in nominal conditions U_n).

1.3.1 Fault Tolerant Control Approaches

Two basics fault tolerant control approaches can be identified, based on the way the control law is redesigned and on the severity of the considered faults.

The first approach is **Passive Fault Tolerance**. It aims at achieving the fault tolerant purpose without changing the control law. This means that both problems $< O, C_n(\theta_n), U >$ and $< O, C_f(\theta_f), U_f >$ ($f \in F$, where F is the set of the considered faults) are solved using the same using the same control law, i.e. they have a common solution. This purpose is generally very difficult to fulfil, and it can be solved only if the objectives require a low level of performances. This approach is similar to the robust methods used in case of system uncertainties. The difference resides first in the size of the changes (i.e. in the difference between fault and uncertainty), and second and more important in the fact that a fault causes the change of the structure of system constraints.

The second approach is called **Active Fault Tolerance**. In this case the system configuration and/or the control law is changed after the fault occurrence. In other words the problems $< O, C_n(\theta_n), U >$ and $< O, C_f(\theta_f), U_f >$ ($f \in F$) have different solutions, i.e. different control laws. To solve this problem generally the knowledge of $C_f(\theta_f)$ and U_f is necessary. This is achieved by the FDI scheme. Depending on the knowledge provided by the FDI method, different strategies of active fault tolerance can be introduced.

Case 1 The FDI algorithm provides the estimates $\hat{C}_f(\hat{\theta}_f)$ and $\hat{U}_f$. Hence the entire control problem $< O, \hat{C}_f(\hat{\theta}_f), \hat{U}_f >$ after the fault is defined.

Case 2 The FDI algorithm provides an estimate $\hat{\Gamma}_f(\hat{\Theta}_f)$ and $\hat{U}_f$ of the fault effect. Here $\hat{\Gamma}_f$ is a set of possible constraints and $\hat{\Theta}_f$ is a set of associated parameters that can describe the system after the fault occurrence. Hence the problem to be solved is the robust control problem $< O, \hat{\Gamma}_f(\hat{\Theta}_f), \hat{U}_f >$. If a solution to this problem exists the controlled system will satisfy the objectives O provided that the actual constraints $C_f(\theta_f) \in \hat{\Gamma}_f(\hat{\Theta}_f)$.

Case 3 The FDI algorithm cannot provide any estimate of the fault effect, i.e. of the model of the system after the fault.

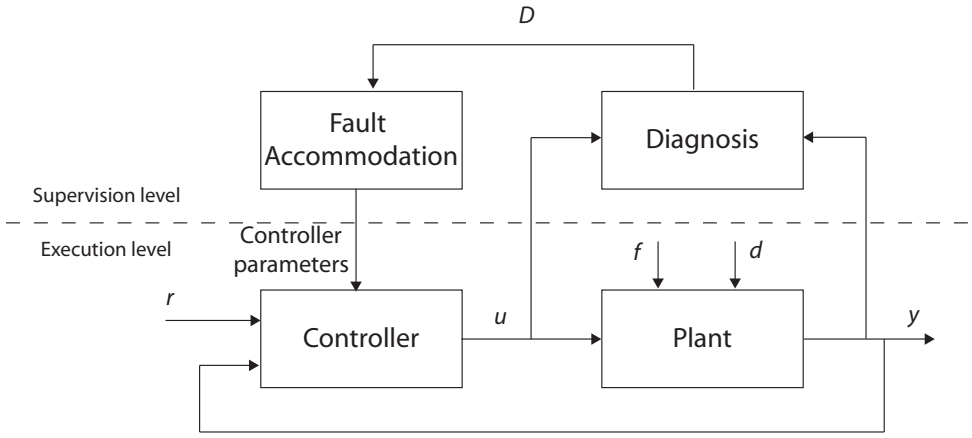


Figure 1.8: Fault Accommodation.

Active fault tolerant strategies are concerned with these three cases. The first strategy is **Fault Accommodation** and it solves the problem for cases 1 and 2. The general architecture for fault accommodation is represented in fig. 1.8: the parameter of the nominal controller are changed to achieve again the system objective after the fault occurrence. At the basis of this strategy there is, then, the claim that it is still possible to control the system with the same kind of controller and achieve the same objectives of the nominal system. In the fault accommodation strategy even **Control Reconfiguration** can be cast (see fig. 1.9). This strategy consists in changing the controller of the system to achieve again the nominal objectives. Practically it is used to solve control problems $< O, \hat{C}_f(\hat{\theta}_f), \hat{U}_f >$ or $< O, \hat{\Gamma}_f(\hat{\Theta}_f), \hat{U}_f >$, which means that:

- the controlled faulty system uses the same components of the nominal one, i.e. no component has been switched off or on in reaction to the fault,
- only the control law is changed to face a different control problem.

The use of architecture in fig. 1.8 or fig. 1.9 depends on the way the fault modifies the system constraints, i.e. on the faulty constraints $C_f(\theta_f)$.

A different strategy is **System Reconfiguration**, which is used to face case 3. In this case part of the faulty system is unknown. For this reason the faulty components are switched off, since they are known from the isolation procedure. The reconfiguration strategy solves the problem $< O, C'_n(\theta_n), U'_n >$, where $C'_n(\theta_n)$ and U'_n are the subsets of constraints and control inputs (respectively) associated with the healthy part of the system. Hence the reconfiguration strategy tries to solve the fault tolerant control problem using only the healthy part of the system, eventually using some healthy components that were not used in nominal conditions.

When it is not possible to satisfy the nominal objectives using fault accommodation or system reconfiguration, an alternative to the fault tolerant solution has to be found. This is the aim of the **Supervision Problem**, which is based on objective reconfiguration. If $\mathcal{O}$ is a set of possible control objectives, the supervision problem consists in finding a solution to $< \mathcal{O}, C(\Theta), U >$ and it is a fault tolerant control problem associated with a decision problem. Usually human operators are involved in the definition of new objectives, but sometimes optimal policies can be used.

In the next subsection some methods to achieve fault tolerance both using fault accommodation and system reconfiguration in linear systems are reported.

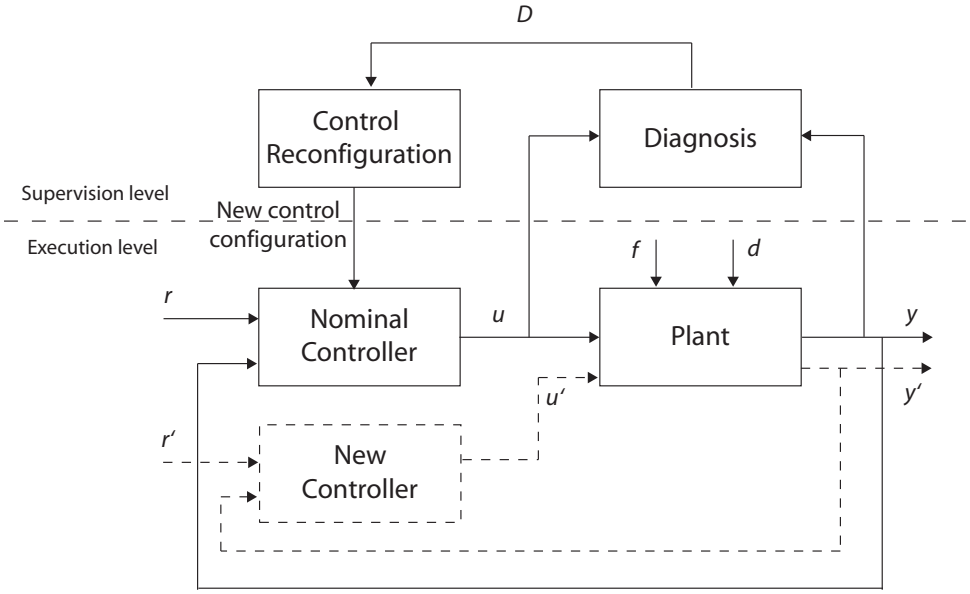


Figure 1.9: Control Reconfiguration.

1.3.2 Fault Tolerant Control Methods

Consider the linear system described by the constraints:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

where $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$.

The first method considered uses **Model Matching** to solve the control problem. The control objective is to design a control law such that the closed-loop system follows the reference model:

$$\dot{x}(t) = Mx(t) + Ne(t)$$

where M, N are given matrices and $e(t)$ is an input signal. The set of admissible control variables is given by the law:

$$u(t) = Ge(t) - Kx(t).$$

The nominal solution to this problem is given by the system:

$$\begin{array}{rcl} A - BK & = & M \\ BG & = & N \end{array} \tag{1.3}$$

with

$$\begin{array}{rcl} Im(A - M) & \subseteq & Im(B) \\ Im(N) & \subseteq & Im(B). \end{array} \tag{1.4}$$

If $rank(B) = m$ the unique solution is given by

$$\begin{array}{rcl} K & = & B^+(A - M) \\ G & = & B^+N \end{array} \tag{1.5}$$

where B^+ is the left pseudo-inverse of B such that $B^+B = I_m$.

If a fault occur and the FDI algorithm estimates the model of the faulty system, i.e. (A_f, B_f), then fault accommodation can be used. However it is not assured that the consistency conditions (1.4) still hold for the faulty system. If they hold the new solution (K_f, G_f) can be computed as the solution of system:

$$\begin{aligned} A_f - B_f K_f &= M \\ B_f G_f &= N \end{aligned}$$

or from the nominal solution as:

$$\begin{aligned} K_f &= [B_f^+ B]K + B_f^+(A_f - A) \\ G_f &= [B_f^+ B]G. \end{aligned}$$

If the consistency conditions do not hold the objective is considered to be not tolerant to the occurred fault, because there is no solution to the system (1.3). However it is always possible to define approximated solutions minimizing the functionals $J_1 = \|A_f - B_f K_f - M\|^2$ and $J_2 = \|B_f G_f - N\|^2$, where $\|\cdot\|$ is the Frobenius norm. With some computation it is possible to check that the solution to this optimal problem is still given by (1.5). The solution of this problem is not reported here, but it can be found in [68], where some stability issues are also addressed.

If the FDI algorithm does not give an estimation of the faulty system (A_f, B_f), but it only gives the faulty components through fault isolation, it is still possible to apply this method using a system reconfiguration policy. In this case the model of the reconfigured system is available, say (A_r, B_r), where some components (the ones affected by the fault) are switched off. Then a solution to (1.3) can still be searched replacing (A, B) with (A_r, B_r).

Another method for fault tolerance in linear systems is to use **Optimal Control**, which is particularly effective for actuator faults. This problem has been addressed in [64] and [65].

The system is modelled by:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ &= Ax(t) + \textstyle\sum_{i\in I} B_i u_i(t) \end{aligned} \tag{1.6}$$

where I is the set of the actuators, $u_i(t) \in \mathbb{R}^{m_i}$ is the input of actuator $i \in I, m = \sum_{i\in I} m_i$, and the pair (A, B) is stabilizable.

The control problem considered has the following specifications:

Objective The system state has to be moved from the initial manifold $(x(t_0), t_0) = (\gamma, 0)$ to the final manifold $(x(t_{fin}), t_{fin}) = (x_{fin}, \infty)$, minimizing the functional

$$J(u, \gamma) = \frac{1}{2}\int_0^\infty [u^\tau(t)Ru(t) + x^\tau(t)Qx(t)]dt. \tag{1.7}$$

Constraints $x(t)$ is a vector of continuous functions of time, $x(t) \in \mathbb{R}^n$, satisfying equation (1.6) $\forall t \geq 0$.

Admissible control signals $u(t) \in U$, continuous functions of time, with values $u(t) \in \mathbb{R}^m$ for open-loop; $u(t) \in U$, continuous functions of the state, with values $u(t) = F(t)x(t)$ for closed-loop.

The solution of this problem in nominal case is given by

$$\begin{aligned} p(t) &= -Kx(t) \\ u(t) &= -R^{-1}B^\tau K x(t) \end{aligned}$$

where K is the solution to the Riccati equation:

$$Q + A^\tau K + KA - KBR^{-1}B^\tau K = 0.$$

Now suppose that at time $t1$ a fault occur such that only a part of the actuators are still healthy. This means that the set of actuators I can be divided in I_{N1} healthy actuators after time $t1$ and I_{F1} faulty actuators after time $t1$, such that $I = I_{N1} \cup I_{F1}$. After the fault the system is described by:

$$\dot{x}(t) = Ax(t) + \sum_{i \in I_{N1}} B_i u_i(t) + \sum_{i \in I_{F1}} \beta_i(u_i(t), \theta_i) \tag{1.8}$$

where $\beta_i(u_i(t), \theta_i)$ describes the system after the fault, hence it is the part to be estimated by the FDI algorithm. The FTC problem aims at solving the optimal control problem with constraints (1.6) for time $t \in [0, t_1[$ and (1.8) for time $t \in [t_1, \infty)$.

Fault accommodation consists in solving the control problem for system

$$\dot{x}(t) = Ax(t) + \sum_{i \in I_{N1}} B_i u_i(t) + \sum_{i \in I_{F1}} \hat{\beta}_i(u_i(t), \hat{\theta}_i) \tag{1.9}$$

instead of system (1.8) after the fault occurrence. In this case $\hat{\beta}_i$ and $\hat{\theta}_i$ are estimated by the FDI algorithm, $\forall i \in I_{F1}$. The solution is computed assuming that the effect of the faulty actuators can be represented by an LTI model:

$$\hat{\beta}_i(u_i(t), \hat{\theta}_i) = \hat{B}_i u_i(t), i \in I_{F1}.$$

The model of the system can be generally described by

$$\dot{x}(t) = Ax(t) + B_1 u(t) \tag{1.10}$$

where $B_1 = (B_{N1}, \hat{B}_{F1}), B_{N1} = (B_i, i \in I_{N1}), \hat{B}_{F1} = (\hat{B}_j, j \in I_{F1})$. Using B_1 the fault accommodation problem can be solved using the nominal optimal control scheme, but a different value of the functional will be obtained, because it will be the sum of the nominal cost before time t_1 with the cost after the fault, i.e. after time t_1.

System reconfiguration in this case consists in solving the control problem for system

$$\dot{x}(t) = Ax(t) + \sum_{i \in I_{N1}} B_i u_i(t) \tag{1.11}$$

after the fault occurrence, i.e. controlling only the healthy part of the system. Even in this case the system can be modelled by (1.10), with $B_1 = (B_{N1}, 0)$.

The problem can be extended to more realistic systems where the faults occur in more different instants of time and admissibility of the solution issues can be addressed. The interested reader is referred to [66] and references therein.

Many other methods have been introduced for control redesign of different kinds of systems. Some of them will be introduced and used in next chapters. For a wider literature on this topic refer to [5].

Chapter 2

Large Scale Systems

Large scale systems are characterized by a large number of variables, nonlinearities and uncertainties (IFAC, Technical Committee on Large Scale Complex Systems). These systems can represent a huge number of modern applications that become more and more complex: power, gas, transportation, manufacturing, water systems, chemical processes, automotive and aeronautic industry are just some of the possible fields in which large scale systems are involved.

There is no definition of large scale systems, because a definition would depend on the way the system is modelled. In fact large scale systems represent modern systems operating in a highly networked environment and for which integration of different technologies and attention to economic, environmental and social aspects are required. The improvement of the technology and of the communication tools makes these systems even more complex and networked, calling for new solutions and skills.

The analysis and synthesis of these systems has to be developed with the use of decomposition techniques, which produce smaller and more manageable subsystems. Moreover there is a need of hierarchical solutions and of coordinating mechanisms able to cope with the time issues of large scale systems.

The recent years have seen the development of many methods for control and analysis of these systems, with the growing attempt to integrate system theory techniques with communication and information techniques. Indeed large scale systems embed control and communication aspects, because the different component of the system are frequently distributed and generally hybrid in their behaviour.

Since the name large scale system has a general meaning and can be associated to different kinds of systems, this chapter will introduce three frameworks in which large scale systems can be better specified.

In section 2.1 distributed system are introduced. With this name networks of computer systems were initially called. In particular the concept of distributed computing started in 1988 at the System Research Center, a laboratory of the Digital Equipment Corporation (now Hewlett-Packard) as a method of computer processing in which different parts of a program run simultaneously on more computers communicating with each other over a network. From distributed computing to distributed systems the step is short, since distributed systems in their first computer framework are viewed as different computers achieving different functions with a main aim and communicating through a network. A survey on this topic with some application issues can be found in [17]. Only recently the properties of distributed systems composed by more general kind of dynamical systems have been studied. In this case distributed systems are composed not only by computers, but also by dynamical subsystems, leading to the possibility of using distributed systems to describe a wider number of applications.

In section 2.2 a modelling formalism is presented for the description of Discrete Event Systems (DES). Nowadays this kind of systems is used to represent a big number of processes when the higher level of system dynamics is the considered level of description. In particular manufacturing systems are described using this formalism. DES are used to represent large scale systems because they are able to model the communication policies and the computer dynamics. However they are not able to catch the dynamics inside each component of the large scale system. A more refined formalism to represent large scale and networked systems is given by Hybrid System, which are described in section 2.3. Hybrid systems represent both the continuous and the discrete dynamics, hence a deeper level of description can be considered and also the subsystems dynamics can be described.

Each of these three representations is a subclass of large scale systems. Moreover DES can be seen as a way of defining distributed systems where only the discrete relations are represented, while hybrid systems can define distributed systems representing also the continuous relations.

2.1 Distributed Systems

Distributed systems are composed by different subsystems, called nodes, connected through a communication network to accomplish a common task. As above mentioned they were born to describe computer systems networks, but their notion can be extended to any kind of network of dynamical systems. Many properties remain the same, only the notation has to be partly changed. In this section first the properties of distributed systems are described in their original computer science connotation. An interesting survey on the topic can be found in [54], while in [43] the real-time problem for distributed systems is faced. Some fault tolerant issues for distributed system are also introduced, basically following the approach in [22].

The remaining of the section is devoted to the extension of the notation to more general classes of distributed systems.

2.1.1 Properties of distributed systems

The main characteristics of a distributed (computer) system are:

- multiple computers;
- interconnections;
- shared state, i.e. the computers cooperate to maintain some global requirements.

The problems to be faced in a distributed system are:

Independent Failure The overall distributed system is preferred to keep working even if one node fails.

Unreliable Communication Messages can be lost due to communication faults, i.e. connections unavailability.

Insecure Communication The interconnection among the computers can be exposed to unauthorized eavesdropping and message modifications.

Costly Communication Lower bandwidth, higher latency and higher cost communication are some drawbacks of the use of many interconnected computers instead of one.

In particular independent failures and costly communication are typical issues of distributed system that usually a centralized system does not have to face.

Nevertheless the main advantage of a distributed architecture is the possibility of sharing information and resources over a wide geographic and organizational spread. Nowadays with the modern technologies and tools it is possible to exploit the distributed nature of these systems to improve the performances of both analysis and synthesis methods (refer to chapter 3). In the following some of the properties that make distributed systems interesting and advantageous from a functional point of view are detailed.

Frequently distributed systems are real-time systems, i.e. systems in which the correctness of their behaviour depends not only on the logical results of the computation but also on the physical instant in which these results are produced, in other words they have to respect a deadline. Real-time systems for which there exists at least one firm deadline that can produce a catastrophe if missed are called hard real-time. A real-time distributed systems is, hence, a set of nodes interconnected by a real-time communication network.

The properties of distributed real-time systems are: composability, scalability and dependability.

Composability ensures that the systems properties follow from subsystem properties. Since large scale systems are built integrating a set of subsystems, an architecture is said to be composable with respect to a given property p if this integration will not invalidate p once it has been established at subsystem level. In a distributed real-time system integration is achieved through the interaction among the different nodes. Therefore the communication system has a central role in determining the composability of a distributed architecture with respect to the temporal properties[1]. There exist two possibilities to implement temporal control:

Event-triggered communication system If a communication system transports event messages the temporal control is external to the communication system, i.e. it is within the sphere of control of the host computers to decide when a message must be sent.

Time-triggered communication system Temporal control resides within the communication system and it is not dependent on the applications in the nodes.

Scalability means that no limits exist to the extensibility of a system and that at the same time complexity of reasoning about the proper operation of any system function do not depend on the system size. Distributed systems usually evolve due to new requirements, new functions added and old functions changed. A scalable architecture is open to these changes, hence there is no limit to its extensibility. Distributed systems provide the necessary framework for scalability:

1. Nodes can be added within the given capacity of the communication channel, introducing additional processing power to the system.

2. If the communication capacity within a cluster[2] is fully utilized, or if the processing power of a node has reached its limit, a node can be transformed into a gateway node to open a way to a new cluster (see 2.1). The interface between the original cluster and the gateway node can remain unchanged.

The property of scalability helps also in controlling the complexity of the distributed system. Complexity is related to the number of parts and the number and type of interactions among

[1] The most stringent temporal demands for real-time systems have their origin in the requirements of the control loop.

[2] A computer cluster is a group of tightly coupled computers that work together closely so that in many respects they can be viewed as though they are a single computer [74].

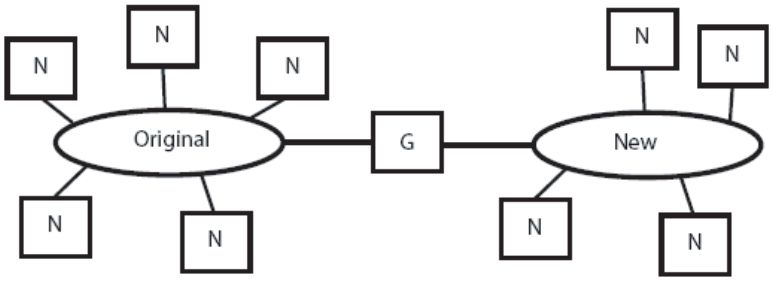


Figure 2.1: Example of the use of gateway nodes for distributed systems scalability.

the parts that must be considered to understand a given function of the system. In a scalable system the effort required to understand any function should remain constant and independent from the system size.

Dependability (see Chapter 1) is the most interesting property for the fault tolerant framework. Indeed thank to this property it is possible to implement well-defined error-containment regions in distributed systems, achieving in this way fault tolerance. Implementing a dependable real-time service requires distribution of functions to achieve fault tolerance.

A fault tolerant system must be structured into partitions that act as error-containment regions so that the consequences of faults that occur in one of this partitions can be detected, corrected or masked before these consequences corrupt the rest of the system. To avoid misunderstanding between the terms fault and error, the error is viewed as the effect of a fault in a computer system. Hence an error-containment region must implement a well-specified service; this service should be provided to the outside world through a small interface, so that an error in the service can be detected at this interface. In a distributed computer system a complete node is regarded as an error-containment region so to perform the error detection at the node's message interface to the communication system.

A tool to implement fault-tolerance in distributed systems is replication. A node must represent a unit of failure (preferably with a simple failure mode), meaning that all inner failure modes of a node are mapped into a single external failure mode. Under this hypothesis node failures can be masked providing replicated nodes which are supposed to show a deterministic behavior (basic concept of replica determinism).

2.1.2 Fault Tolerance in Distributed Systems

Fault tolerance is distributed systems is strictly related to their property of dependability. It can be implemented following two ways:

- At the architectural level, transparent to the application code. This type of tolerance is called **systematic fault tolerance**: the architecture must provide replica determinism so that fault tolerance can be achieved by the temporal or spatial replication of computations to detect and mask the faults specified in the fault hypothesis.

- At the application level, within the application code. We will call this type of fault tolerance **application-specific fault tolerance**: it intertwines the normal processing functions with the error-detection and fault-tolerance functions at the application level.

An error is a discrepancy between the intended correct state and the current state of a system. The goal of the fault-tolerant system is to detect and mask or repair errors before they show up as failures at the system user service interface. Error detection requires that, along with the information about the current state, knowledge about the nominal state of a system is available.

The more is known a priori about the properties of correct states and the temporal patterns of correct behavior of a computation, the more effective are the error detection techniques. Techniques for error detection based on a priori knowledge are:

Syntactic knowledge about the code space : parity bits, error-detecting codes in memory, cyclic redundancy check (CRC) in data transmission, check digits at the man-machine interface. Such codes are very effective in detecting the corruption of a value stored in memory or in the transmission of a value over a computer network.

Assertions and acceptance tests : application specific knowledge about the restricted ranges and the known interrelationship of the values of the entities can be used to detect additional errors that are undetectable by syntactic methods.

Activation patterns of computations : knowledge about the regularity in the activation pattern of a computation can be used to detect errors in the temporal domain.

Worst case execution time of tasks : it can be used to detect task errors in the temporal domain.

On the other hands there are many different possible combination of hardware, software and time redundancy (see Chapter 1) that can be used to detect different types of errors by performing the computation twice.

2.1.3 Distributed Systems: a General Description

The notion of distributed systems is well known in the control community from the so called *distributed control systems* introduced in 1975 by Honeywell and Yokogawa [75]. These systems are basically systems for which the controller is not centralised, but is distributed throughout the system, i.e each component sub-system is under the control of one or more controllers. The overall system is generally networked, to achieve communication and monitoring. This structure is used in many applications, including industrial, electrical, computer and civil engineering, with manufacturing and process systems.

From this first notion an extension has been introduced with the development of functional architectures (see Chapter 3) and the standard IEC 61499 (see [46]) for functional decomposition of distributed systems.

The basic way in which distributed dynamical systems are represented is in the form of networks, although there is some kind of difference between a distributed system and a network of systems. As stated in [72] for computer systems, the basic difference resides in who is responsible for moving the data around the system: if it is the system then distributed systems are invoked, if it is the used it is more appropriate to talk about networks of systems. Basically in a distributed system a collection of independent subsystems appears to the user as a single coherent system, while in a network this coherence is not transparent to the user, even if there is some.

Beyond this confusion between distributed and networked systems, in general these systems are viewed in the control community as a set of agents aimed at achieving some specified collective goal, by coordinating and/or competing with each other.

Even if this kind of systems has been introduced quite early in time, there exists a few control literature on the topic. This is due to a huge effort in finding a common framework to

deal with them. Basically the properties and the methods can be borrowed from the computer definition of distributed systems and extended to the dynamical systems in the control theory. However some attention must be paid when trying to extend the fault tolerant notions. Indeed in computer systems the faults usually develop as errors in the machines, while in general distributed dynamical systems their effect is called failure and it is viewed and handled as a loss of function, as it will be better stated in Chapter 3.

2.2 Discrete Event Systems

In this section a brief introduction to Discrete Event Systems (DES) is reported. All the results presented in this section are based on [18].

As mentioned above, DES can model many characteristics of distributed systems, hence they are a good candidate to study the properties of a given distributed system, especially for what concerns fault tolerance.

A DES is a dynamical system where the state space is described by a discrete set, state transitions are only observed at discrete points in time and are associated with *events*. An event can be identified with a specific action taken, either spontaneous (dictated by nature) or as a result of some conditions which are suddenly met. If e is an event, a system can be affected by different types of events contained in the an event set E.

Definition 2.1. Discrete Event Systems
A discrete event system is a discrete-state, event-driven system, i.e. its state depends entirely on the occurrence of asynchronous discrete events over time.

Des are usually described using automata and languages. Languages describing DES represent them in terms of sequences of events. A language can be:

- **Timed** if it represents the set of all the possible temporal sequences of events a given DES can execute;
- **Untimed** if it represents simply the set of all the possible sequences of events a DES can execute;
- **Stochastic Timed** if the DES is stochastic.

This representation gives rise to three level of abstraction in the study of DES:

1. *Logic* if only the logic behaviour is of interest, i.e. the order in which the events take place. At this level an untimed language is used.
2. *Timed* if also the instant of time in which the events take place is of interest. At this level a timed language is necessary.
3. *Stochastic* if the considered DES has a stochastic behaviour in the event sequence, thus it need a stochastic timed language to be described.

The focus in this overview will be on untimed languages and automata. To each DES it is possible to associate an event set E. This is the alphabet of a language and event sequences are strings (words) in that language. A string consisting of no events is called empty string and is denoted by ϵ. The length of a string s, denoted with $|s|$, is the number of events contained in it, counting multiple occurrences of the same event.

Definition 2.2. Language
A language defined over a set E is a set of finite-length strings of events in E.

Languages are built from a set of events E using concatenation of strings: the string abb is the concatenation of the string ab and the event b. The empty string ϵ is the identity element of concatenation. Let E^* denote the set of all finite strings of elements of E, including the empty string ϵ; the $(\cdot)^*$ operation is called Kleene-closure. A language over an event set E is a subset of E^*.

For a string $s = tuv$ it is defined: t prefix of s, u substring of s, v suffix of s. Both ϵ and s are prefixes, substrings and suffixes of s.

The usual set operations, such as union, intersection, difference and complement with respect to E^* are applicable to languages, since languages are sets. In addition it is possible to introduce the following operations:

Concatenation Let $L_a, L_b \subseteq E^*$, then

$$L_a L_b := \{s \in E^* : (s = s_a s_b) \text{ and } (s_a \in L_a) \text{ and } (s_b \in L_b)\}.$$

A string is in $L_a L_b$ if it can be written as the concatenation of a string in L_a with a string in L_b.

Prefix-closure Let $L \subseteq E^*$, then

$$\overline{L} := \{s \in E^* : \exists t \in E^* (st \in L)\}.$$

The prefix closure of L is the language $\overline{L}$ consisting of all the prefixes of all the strings in L. In general $L \subseteq \overline{L}$.

Kleene-closure Let $L \subseteq E^*$, then

$$L^* := \{\epsilon\} \cup L \cup LL \cup LLL \cup \cdots .$$

An element of L^* is formed by the concatenation of a finite number of elements of L.

2.2.1 Automata

Automata are a tool to describe a language and thus DES.

Definition 2.3. Automaton
A deterministic automaton G is a six-tuple

$$G = (X, E, f, \Gamma, x_0, X_m)$$

where

- X is the set of states;
- E is the finite set of events associated with transitions in G;
- $f : X \times E \to X$ is the transition function representing the transition from one state to another due to a given event;
- $\Gamma : X \to 2^E$ is the active event function, i.e. $\Gamma(x)$ is the set of all the events e for which $f(x, e)$ is defined;

- x_0 is the initial state;
- $X_m \subseteq X$ is the set of marked states.

The language generated by G is

$$\mathcal{L}(G) := \{s \in E^* : f(x_0, s) \text{ is defined}\}$$

The language marked by G is

$$\mathcal{L}_m(G) := \{s \in E^* : f(x_0, s) \in X_m\}$$

Two automata are said to be equivalent if they generate and mark the same languages.

An automaton G is said to be blocking if

$$\overline{\mathcal{L}_m(G)} \subset \mathcal{L}(G)$$

and nonblocking when

$$\overline{\mathcal{L}_m(G)} = \mathcal{L}(G).$$

Operations on automata are: accessible part (Ac), coaccessible part ($CoAc$), trim ($Trim$), complement ($(\cdot)^{comp}$), product ($(\cdot) \times (\cdot)$) and parallel composition ($(\cdot) \parallel (\cdot)$). The interested reader is referred to [18] for a deeper description of these operations. An interesting operation which will be used throughout this work is *parallel composition*. Consider two automata $G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1})$ and $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$. The parallel composition of G_1 and G_2 is the automaton:

$$G_1 \parallel G_2 = Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_{1\parallel 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

where

$$f((x_1, x_2), e) = \begin{cases} (f_1(x_1, e), f_2(x_2, e)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1, e), x_2) & \text{if } e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1, f_2(x_2, e)) & \text{if } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

In parallel composition a common event can only be executed if both the two automata execute it simultaneously. The two automata are synchronized on common events.

A nondeterministic automaton G_{nd} is a six-tuple

$$G_{nd} = (X, E \cup \{\epsilon\}, f_{nd}, \Gamma, x_0, X_m)$$

where

- $f_{nd} : X \times E \cup \{\epsilon\} \to 2^X$, such that $f_{nd}(x, e) \subseteq X$ whenever it is defined;
- the initial state $x_0 \subseteq X$ may be a set of states.

It is always possible transform a nondeterministic automaton G_{nd} into an equivalent deterministic one. We will call the resulting equivalent deterministic automaton the observer G_{obs} corresponding to the nondeterministic automaton. The languages generated and marked by the nondeterministic automaton and its observer are the same. Hence if an automaton has unobservable events it is possible to build a nondeterministic automaton substituting all the unobservable events with ϵ-transitions and then design an observer for it.

A language is said to be regular if it can be marked by a finite-state automaton. The class of regular languages is $\mathcal{R}$.

2.2.2 Supervisory control of DES

Consider an automaton G that models the uncontrolled behaviour of the DES. This behaviour is not satisfactory because the system has to follow some specifications. A controller has restrict its behaviour to a subset of $L(G)$. In this framework, we consider sublanguages of $L(G)$ that represent the desired behaviour for the controlled system and are called admissible languages. In this paradigm, the supervisor S observes some of the events that G executes and tells G which events in its current active set are allowed next. In other words, S has the capability of disabling some feasible events of G, exerting in this way a feedback control action on G.

Consider a DES modelled by the automaton $G = (X, E, f, \Gamma, x_0, X_m)$ generating language L and marking language L_m. L and L_m are defined over the event set E. Consider the case of L prefix-closed. Let E be partitioned into two disjoint subsets: $E = E_c \cup E_{uc}$, where E_c is the set of controllable events, i.e. the events that can be disabled by a supervisor, and E_{uc} is the set of uncontrollable events. Assuming that all the events in E are observable, the supervisor E can be described by the function

$$S : \mathcal{L}(G) \to 2^E$$

such that for each $s \in \mathcal{L}(G)$

$$S(s) \cap \Gamma(f(x_0, s))$$

is the set of enabled events the G can execute in its current state $f(x_0, s)$. Hence S is said to be admissible if $\forall s \in \mathcal{L}(G)$

$$E_{uc} \cap \Gamma(f(x_0, s)) \subseteq S(s)$$

or in words: S is not allowed to disable a feasible uncontrollable event.

The automaton G supervised by S is named S/G. The language generated by S/G is defined recursively as:

1. $\epsilon \in \mathcal{L}(S/G)$
2. $[(s \in \mathcal{L}(S/G) \text{ and } (s\sigma \in \mathcal{L}(G)) \text{ and } (\sigma \in S(s))] \iff [(s\sigma \in \mathcal{L}(S/G))]$

The language marked by S/G is defined as

$$\mathcal{L}_m(S/G) := \mathcal{L}(S/G) \cap \mathcal{L}_m(G).$$

In the following some theorems to deal with uncontrollability are reported. For the proofs refer to [18].

Theorem 2.1. Controllability Theorem
Consider a DES $G = (X, E, f, \Gamma, x_0)$ where $E_{uc} \subseteq E$ is the set of uncontrollable events. Let $K \subseteq \mathcal{L}(G)$, where $K \neq \emptyset$ is the admissible language. Then there exists a supervisor S such that $\mathcal{L}(S/G) = K$ is and only if the controllability condition holds, i.e.

$$\overline{K} E_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}.$$

Definition 2.4. Controllability
Let K and $M = \overline{M}$ be languages over set E. Let E_{uc} be a subset of E. K is said to be controllable with respect to M and E_{uc} if and only if

$$\overline{K} E_{uc} \cap M \subseteq \overline{K}.$$

Theorem 2.2. Nonblocking Controllability Theorem
Consider a DES $G = (X, E, f, \Gamma, x_0)$ where $E_{uc} \subseteq E$ is the set of uncontrollable events. Consider the language $K \subseteq \mathcal{L}_m(G)$, where $K \neq \emptyset$ is the admissible language. Then there exists a nonblocking supervisor S for G such that $\mathcal{L}_m(S/G) = K$ and $\mathcal{L}(S/G) = \overline{K}$ if and only if

1. K is controllable with respect to $\mathcal{L}(G)$ and E_{uc}, i.e.
$$\overline{K}E_{uc} \cap \mathcal{L}(G) \subseteq \overline{K},$$
2. K is $\mathcal{L}_m(G)$-closed, i.e.
$$K = \overline{K} \cap \mathcal{L}_m(G).$$

Now let assume that language $K \subseteq \mathcal{L}(G)$ is controllable, then from controllability theorem the supervisor S is defined as

$$S(s) = [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \overline{K}\}$$

hence $\mathcal{L}(S/G) = \overline{K}$. To design the function S consider automaton $R = (Y, E, g, \Gamma_R, y_0, Y)$ which marks language $\overline{K}$. R is trim and $\mathcal{L}_m(R) = \mathcal{L}(R) = \overline{K}$. If R and G are connected by the product operation, then the result $R \times G$ is exactly the required behaviour for S/G:

$$\begin{aligned}
\mathcal{L}(R \times G) &= \mathcal{L}(R) \cap \mathcal{L}(G) \\
&= \overline{K} \cap \mathcal{L}(G) \\
&= \overline{K} = \mathcal{L}(S/G) \\
\mathcal{L}_m(R \times G) &= \mathcal{L}_m(R) \cap \mathcal{L}_m(G) \\
&= \overline{K} \cap \mathcal{L}_m(G) \\
&= \mathcal{L}(S/G) \cap \mathcal{L}_m(G) = \mathcal{L}(S/G).
\end{aligned}$$

Since R is defined to have the same event set as G, then $R \parallel G = R \times G$. R is called the standard realization of S.

If not all the events of G are observable, E can be partitioned as $E = E_o \cup E_{uo}$, where E_o is the set of observable events and E_{uo} is the set of unobservable events. In this case the feedback loop includes a natural projection P (see [18]) between G and the supervisor S. If the supervisor cannot distinguish between two strings s_1 and s_2 that have the same projection the same control action is performed: $S_P[P(s_1)] = S_P[P(s_2)]$. Hence a partial observation supervisor is defined as the function $S_P : P[\mathcal{L}(G)] \to 2^E$. This means that the control action can change only after the occurrence of an observable events, i.e. when $P(s)$ changes. For the unobservability problem analogous results exist to the ones for the controllability problem. For these results and for admissibility of S_P refer to [18] and reference therein.

2.3 Hybrid Systems

Hybrid systems are frequently used to describe networked systems in a deeper level than DES. Indeed when the nodes description is also required, hybrid systems are able to cope with both the discrete and the continuous nature of distributed systems. This is due to the nature of Hybrid Systems: they are systems in which continuous and discrete dynamics interact. Hence with the hybrid representation it is possible to describe both the behaviour of the network and the behaviour of each node.

In this section some techniques for modelling hybrid systems are reported. The fault detection issues will be addressed in Chapter 4.

As stated in [2] two main modelling frameworks have been established for hybrid systems. These frameworks follow two different conceptions related to the nature of these systems:

1. The first is based on the extension of continuous dynamics models to represent discrete dynamics too. This is achieved by including discrete time and variables to ordinary differential equations to describe jumps and switchings. The approaches that follow this strategy are basically able to deal with complex continuous dynamics and emphasize stability results.

2. A different approach is embedded in computer science models and aims at extending the verification methodologies. In particular the methods cast in this framework modify the discrete dynamics models to deal with continuous dynamics too. These techniques are able to face with complex discrete dynamics (finite automata) and emphasize analysis results (verification) and simulation methods.

The right choice of the model and of the class to represent hybrid systems is in general achieved based on the system property under study, and obviously very application dependent.

One of the first attempts to find a unified methodology for modelling hybrid systems has been presented in [9] in 1998. A classification of hybrid phenomena is introduced to better find a common model to represent them. The continuous dynamics of a hybrid system are modelled by:

$$\dot{x}(t) = \xi(t), t \geq 0 \tag{2.1}$$

depending on discrete phenomena. Here $x(t)$ is the continuous component of the state, $\xi(t)$ is a controller vector field generally depending on $x(t)$ and on $u(t)$, the continuous component of the control policy. The discrete phenomena that influence system 2.1 are:

1. Autonomous Switching: they occur because the vector field $\xi(t)$ changes discontinuously, or switches, when the state $x(t)$ hits certain boundaries.

2. Autonomous Impulses: the continuous state $x(t)$ changes impulsively on hitting prescribed regions of the state space.

3. Controlled Switching: $\xi(t)$ switches in response to a control command with an associated cost, implying switches between different vector fields.

4. Controlled Impulses: $x(t)$ jumps in response to a control command with an associated cost.

Based on this classification of hybrid phenomena it is possible to define different model for hybrid systems, as described in [9]. The authors cast all these models in a general description, given by the Controlled General Hybrid Dynamical System (CGHDS)

$$H_c = [Q, \Sigma, A, G, V, C, F]$$

where

- Q is the set of discrete states;
- $\Sigma = \{\Sigma_q\}_{q \in Q}$ is the collection of controlled dynamical systems, where $\Sigma_q = [X_q, \Gamma_q, f_q, U_q]$ is a controlled dynamical systems, X_q is the continuous state space, Γ_q is the transition semigroup, f_q represents the continuous dynamics and U_q is the set of continuous control;
- $A = \{A_q\}_{q \in Q}, A_q \subset X_q$, for each $q \in Q$, is the collection of the autonomous jump sets;
- $G = \{G_q\}_{q \in Q}, G_q : A_q \times V_q \rightarrow S$ is the autonomous jump transition map, parameterized by the transition control set V_q such that $V = \{V_q\}_{q \in Q}$.
- $C = \{C_q\}_{q \in Q}, C_q \subset X_q$ is the set of controlled jump sets;

- $F = \{F_q\}_{q \in Q}, F_q : C_q \to 2^S$ is the set of controlled jump destination maps.

The above defined system can be represented by an automaton as described in [9]. Hence it can be considered in between the first and the second of the above described frameworks.

Note that the control problem is not referred here, because it is out of the aim of this work. The interested reader can refer to [3] and reference therein.

2.3.1 A control oriented modelling framework

For the first modelling framework, an interesting method has been introduced in [3]. Since a hierarchical vision of hybrid systems is not always possible, with the continuous dynamics controlled at lower level and the discrete dynamics supervised at higher level, mixed logical dynamical (MLD) systems are introduced. These systems are able to model interacting physical laws, logical rules and operating constraints. Their description is based on propositional calculus and linear integer programming. A MLD system is given by the equations:

$$\begin{aligned} x(t+1) &= A_t x(t) + B_{1t} u(t) + B_{2t} d(t) + B_{3t} z(t) \\ y(t) &= C_t x(t) + D_{1t} u(t) + D_{2t} d(t) + D_{3t} z(t) \\ E_{2t} d(t) + E_{3t} z(t) &\leq E_{1t} u(t) + E_{4t} x(t) + E_{5t} \end{aligned}$$

where $t \in \mathbb{Z}$;

$$x = \begin{bmatrix} x_c \\ x_l \end{bmatrix}, x_c \in \mathbb{R}^{n_c}, x_l \in \{0,1\}^{n_l}, n = n_c + n_l$$

is the state of the system, whose components are distinguished between continuous x_c and binary x_l;

$$y = \begin{bmatrix} y_c \\ y_l \end{bmatrix}, y_c \in \mathbb{R}^{p_c}, y_l \in \{0,1\}^{p_l}, p = p_c + p_l$$

is the output vector;

$$u = \begin{bmatrix} u_c \\ u_l \end{bmatrix}, u_c \in \mathbb{R}^{m_c}, u_l \in \{0,1\}^{m_l}, m = m_c + m_l$$

is the command input, collecting both continuous commands u_c, and binary (on/off) commands u_l; $\delta \in \{0,1\}^{r_l}$ and $z \in \mathbb{R}^{r_c}$ represent respectively auxiliary logical and continuous variables.

These systems are able, with some manipulations, to represent many kinds of different systems: piecewise linear and affine, discrete inputs and qualitative outputs, bilinear systems and automata.

In [36] the equivalence among different classes of hybrid systems is established: mixed logical dynamical systems, linear complementarity (LC) systems, extended linear complementarity (ELC) systems, piecewise affine (PWA) systems, and max-min-plus scaling (MMPS) systems. Each of these classes has its own characteristics in terms of problems that can be solved. PWA systems are generally used to solve stability problems and to set stability criteria, estimation and identification techniques. MLD systems are usually a good basis to solve control and verification problems. MMPS systems are also used to design control techniques, while LC systems have been used to test conditions on existence and uniqueness of solutions.

2.3.2 A discrete event modelling framework

More oriented to the second framework is the theory of hybrid automata. They represent a mathematically concrete language, rich enough to demonstrate many issues, but simple enough

to do it without excessive mathematical formalism and notation (see [41]). An autonomous hybrid automaton is defined as

$$H = (Q, X, f, Init, D, E, G, R)$$

where

- Q is a discrete state space;
- $X = \mathbb{R}^n$ is a continuous state space;
- $f : Q \times X \to \mathbb{R}^n$ is a vector field;
- $Init \subset Q \times X$ is a set of initial states;
- $D : Q \to P(X)$[3] is a domain;
- $E \subset Q \times Q$ is a set of edges;
- $G : E \to P(X)$ is a guard condition;
- $R : E \times X \to P(X)$ is a reset map.

$(q, x) \in Q \times X$ is the state of H. The behaviour of a hybrid automaton can be described in words as: starting from an initial value $(q_0, x_0) \in Init$, the continuous state x flows according to the vector field $f(q_0, \cdot)$, while the discrete state q remains constant as long as x remains in $D(q_0)$; if x reaches a guard $G(q_0, q_1)$, for some $(q_0, q_1) \in E$, the discrete state changes to q_1; simultaneously the continuous state resets to some value in $R(q_0, q_1, x)$; after the discrete transition, continuous evolution resumes and the whole process is repeated. Hybrid automata have a visual interpretation as directed graphs with vertices Q and edges E.

Hybrid Input/Output Automata (HIOA) have been described in [50]. A HIOA is given by

$$HA = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, D, W)$$

where:

- U, X, Y are the variables of the system, respectively input, internal and output variables; they are continuous and their union gives the set of variables $V = U \bigcup X \bigcup Y$;
- $\Sigma^{in}, \Sigma^{int}, \Sigma^{out}$ are the actions of the system, respectively input, internal and output actions; they are discrete and their union gives the set of actions $\Sigma = \Sigma^{in} \bigcup \Sigma^{int} \bigcup \Sigma^{out}$;
- Θ is the set of initial states;
- D is the set of discrete transitions;
- W is the set of trajectories.

The actions are what in discrete event systems are also called events, i.e. discrete changes in the system, while the transitions represent the evolution of the discrete state, i.e. when the system goes from one discrete state to another this means that a transition has occurred. In other words actions lead to discrete (atomic) transitions from a state to another. In the same way the variables describe a continuous behavior which is represented by the trajectories of the system. In this framework an execution of the HIOA is a sequence of trajectories and actions.

[3] $P(X)$ denotes the power set (set of all subsets) of X.

Chapter 3

Distributed Fault Tolerant Systems

This chapter presents a way of dealing with fault tolerance in distributed systems. At a very high level of abstraction the system is not analysed following the properties of a particular modelling formalism. Fault tolerance is then obtained building an architecture of diagnosis and supervision that is able to cope with the nature of the system. This approach grounds on decentralised policies to manage fault detection and reconfiguration at different layers in the system.

Note that the presented approach does not introduce any new fault detection and reconfiguration method. The novelty of the procedure resides in the way the architecture is designed and in the application of decomposition methods to distributed systems. Another feature of this procedure is the decision policy: diagnostic signals and reconfiguration possibilities are sent throughout the system and analysed at different levels to decide if something has occurred.

In the fault detection field, centralised methods are usually applied to dynamical systems and well known in literature. Nevertheless decentralised methods seem to be more suitable for distributed systems because they follow the nature of these systems, applying the ancient method of the Roman Senate "divide et impera". When fault detection and fault tolerance problems are faced, the decentralisation of the detection task is improving the possibility of stopping the effects of the fault before they reach a vital part of the system. Moreover if the considered systems are large and complex, many different components are interconnected and the centralised approaches generally require too much global knowledge, leading to a big computational burden.

Applying a decentralised policy to a distributed system means finding a way to decompose the system in its subsystems. This decomposition should lead to a practical architecture able to enlighten the detectable and reconfigurable parts of the system inside an "error"containment region. The first part of the decentralising process should then be dedicated to the modelling of the system, meaning to find an abstraction of the system which can be easily used to apply fault tolerance methods. An approach in this sense for general distributed systems has been introduced in [24], where the system is modelled as a collection of components (or agents) taking the decisions. More related to the diagnostic problem, the method proposed in [59] deals with large scale discrete event systems (such as telecommunication networks) and highlights how the decentralised methods are the easiest way to model distributed systems. A somehow different reasoning is presented in [61]: the author presents a way of dealing not only with cooperating components (which is the usual way distributed systems are described), but also with competing components. In this way some drawbacks of decentralised systems are also highlighted.

All these methods generally are applied to computer systems, i.e. distributed system where the nodes are computers. The aim of the approach introduced in Sec. 3.3 is to design a decentralised architecture for the most general kind of distributed system, including computers, but also devices and operators.

In this chapter the decomposition methods presented are aimed to design an architecture for fault tolerance based on the functional description of a given distributed system. To this aim, in the next section functional and structural decomposition policies are introduced, and an algorithm to obtain a functional model from a structural one is presented. The algorithm allows the designer to extrapolate the functions of a system in a unique and procedural way, starting from the structural properties of the system and has been presented in [16] and in [14]. This method will be used in section 3.3 to design the fault tolerant architecture (see [6], starting from the general framework proposed in section 3.2 (first publications in this direction have been [10], [13] and [12]). Finally the procedure is applied to an example with experimental results (see [11] and [15].

3.1 System Decomposition Policies

At the basis of a decentralised strategy, wether it is for fault tolerance or for any other purpose, there is the decomposition method used to find out or only express the different parts of a system.

In particular the decomposition procedure must be able to highlight the differences among the parts composing the system, in terms of particular distinguishing features. This is obviously easier when talking about distributed systems, which, by definition, are composed by nodes with differentiating characteristics, mostly in the way they act on the system global task achievement.

Due to this nature of distributed systems, a decomposition procedure for them should point out the tasks each node is performing. If the final aim of the decomposition procedure is fault tolerance, the nodes of the system can be further decomposed in order to simplify the fault management.

To this end in next subsection functional decomposition procedures are presented, which are able to represent the system in terms of the task accomplished by its composing parts. In the following subsections it will be also shown how the analysis of the structural properties of a system can lead to clarify the internal distribution of system tasks.

3.1.1 Functional Abstraction of System Tasks

Functional reasoning has been introduced to cope with the necessity of engineers to deal with complex systems composed by many interconnected components. Indeed the so called Process (or Piping) and Instrumentation Diagrams (P&ID) and other block diagram models are used by process engineers to point out the different functional parts of the considered system, in terms of components of the system itself. This first hardware decomposition is then extended with a software implementation by the concepts established by the standard IEC 61499 (see [46]), based on the software developed in the previous standard IEC 61131-3. The novelty of this new standard resides in its ability to deal with distributed (control) systems. Beyond the software implementation, the theoretical results on which this standard is based are well represented by other functional methods to highlight the functions of a system, such as functional trees and Generic Component Models (see [69] and [70]).

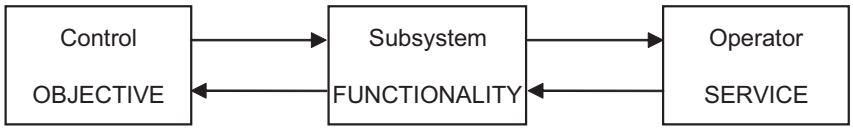


Figure 3.1: Different points of view on the meaning of function.

The functional analysis is based on the concept of **function**, or functionality. This concept is quite general, because it depends on the point of view of the user. In fig. 3.1 a scheme is reported, representing the different names of "function"referred to different points of view: from the controller point of view the function is called objective, and it is related to the aim of the control law; from the operator point of view the function is represented by a service the system has to provide to accomplish the task it is demanded; finally what it called functionality is the aim for which a certain system/component is assembled to. Abstracting the mathematical definition of a function as the equations representing a certain object and the physical definition of a function as the results of some physical constraints, a function (or service, or objective, or functionality) represents the behaviour of the system/subsystem/component to accomplish a certain desired task. The function then depends generally on the aim for which a certain object is chosen and on the interconnection with the other components. This immediately recalls the description of the nodes of distributed systems (see Chapter 2). That is the reason why functional analysis has been referred to as the most suitable tool for the study of distributed systems and their characteristics.

Even without forcing a fairly impossible objective definition of "function", some specifications for the system functional decomposition will be helpful in fully understand how a function can be detected. The decomposition always depends on the tasks the user wants the system and its parts to accomplish. Another choice to be made is the level of deepness of the decomposition, i.e. on how deeply we want the system to be decomposed and in how many parts. At the ground level there are the functions associated to the components of the system, such as actuators, sensors, etc. Aggregation of these components leads to higher level components, also called subsystems, which are meant to perform some tasks related to the ground functions through the interconnections.

From now on the terms function, functionality, service and objective will be used indifferently with the above explained meaning.

Functional methods are founded on the concepts of the standard IEC 61499, where the notion of functional block is introduced to deal with complex distributed control systems. The necessity to create this standard is due to the increasing use of object oriented languages. In fact it aims at decomposing the system in its basic functions, represented by functional block which can be composed to fulfil a general purpose. One of the features of this policy is reusability, because the functional blocks can be collected in libraries and be reused in other applications.

This is obviously a software implementation of the functional concepts. Anyway the same idea has been exploited to build Generic Component Models (GCM) in a more control oriented version. This method is used to state a relation between properties on system variables and services. Assume a given subsystem, which is composed by the interconnection of a set of components, is in charge of providing some functionality, for example a set of sensors S provides a set of signals y by which a vector z is estimated. Let the notation $O(z/S)$ stand for the property "z is observable by the set of sensors S". $O(z/S)$ is obviously a necessary condition for the estimation service to exist. For each service σ provided by a subsystem C let z be a functional of the state and let P be a property such that

$$P\,(z/C) \implies \sigma \text{ is available.}$$

Let $\Gamma \subseteq C$ be a subset of components. Let 2^{C^+} be the collection of all subsets of C such that

$$P\,(z/\Gamma) \iff \Gamma \in 2^{C^+}.$$

Therefore, there is a possibility of providing the service σ by using any subset of components

$\Gamma \in 2^{C^+}$, which means that different *versions* σ_Γ of the service can be designed, using different *resources* Γ.

Let

$$N(\sigma) = \left|2^{C^+}\right|$$

then, there are $N(\sigma)$ different versions, that constitute a set $v[\sigma]$. Each version of that set is able to provide the service σ. Since only one can be run at time t, they must be ranked using an appropriate procedure (see [69] and [71] for details and comments).

A tool to obtain functional graphs of a system is the *Functional Tree* (see [1]). The global function of the system (root of the tree) is expressed in terms of relations with and among sub-functions. In particular the functional AND tree is a classical representation of services hierarchy. It is a hierarchical graph where nodes are services, and edges exist between a node σ and its followers $\sigma' \in F(\sigma)$ at the next lower level if and only if the availability of service σ needs - and is implied by - the availability of all of the services in $F(\sigma)$. The functional tree represents necessary and sufficient relationships among functionalities, i.e. necessary and sufficient conditions for the properties on the variables to be satisfied. For this reason it is a powerful tool for control and analysis of distributed systems, since it allows the designer to consider the different levels of the tree independently and then exploit the distributed nature of the system, for example controlling each level independently or for FDI and Fault Tolerance considering the effects of faults at each level. A top-down analysis of the tree shows the necessary conditions for a higher level property to be achieved, each level of the tree reports the necessary functionalities for the immediately higher level functionality to be fulfilled. A bottom-up analysis shows the sufficient conditions for a functionality to be reached. Note that in this case for each node σ all its followers $\sigma' \in F(\sigma)$ must be considered, because of the AND operator that applies to them. Designing the functional tree can therefore be done by starting from the top functionality and finding all the necessary conditions for it to be available. These conditions can be very easily linked to properties that the controlled system must fulfill.

Now assume that at time t there is only a subset $\Gamma(t) \subset C$ of healthy components available, as the consequence of previously occurred faults. The functionality σ is available as long as $\Gamma(t)$ contains at least one subset that belongs to 2^{C^+}, i.e. as long as the set of versions $v[\sigma]$ is not empty. Let $N(\sigma, t)$ be the number of versions of σ available at time t (this number is called the redundancy degree of service σ). Since each node σ in the AND tree represents a service which at time t can be provided under $N(\sigma, t)$ versions, the AND functional tree can be replaced by an AND/OR functional tree, where AND and OR nodes alternate: AND nodes display the lower level services that are all necessary for a higher level service to be available, and for each service, an OR node displays the different exclusive versions that are available to perform it. This means that the AND/OR functional tree introduces the redundancy relations between components. For example if σ_1 and σ_2 are two nodes of the tree connected with an OR gate to the upper level node σ' this means that σ_1 and σ_2 are two different versions of services through which it is possible to obtain σ'. As a consequence, when faults occur such that version σ_1 is no longer available, the system operation can be continued thank to version σ_2.

The *Fault Tree* is another graphical tool showing the map of loss of functions as a consequence of faulty conditions (see [1], [5]). The fault tree is related to the Functional Tree using complementarity reasoning. Losses of functions at each level of the tree are seen as caused by losses of functions at lower levels until the main elementary cause, given by the physical fault, is reached. The fault tree is generally used to study the possibilities of detection of losses of functions. The detectable losses of functions are called **failure modes**.

An example of Functional and Fault Trees and their relation is reported in fig. 3.2 where

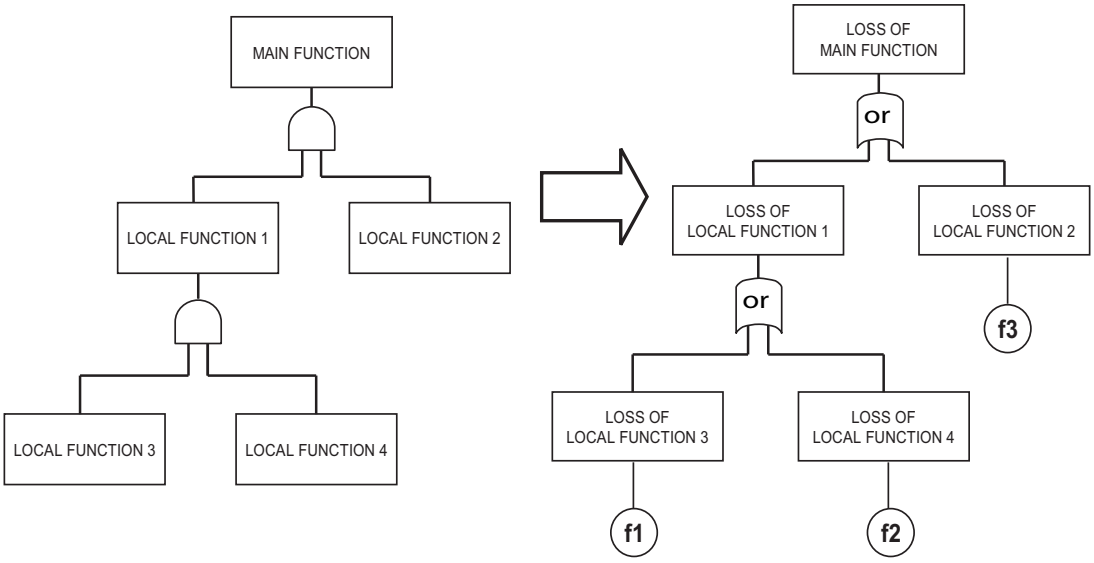


Figure 3.2: Functional and fault tree: complementarity.

the complementarity of the design is highlighted. In Sec. 3.3 the use of these tools for failure diagnosis and reconfiguration will be shown.

3.1.2 Analysis of System Structural Properties

The structural model of a system is a qualitative representation of its behaviour. This means that it is objective because it represents the physical constraints that apply to the system components, and it is also easy to obtain, because it needs no parameter identification. Practically the structural model represents the relations between the system variables imposed by the system components. These relations are represented through a bi-partite graph, called structural graph, independent of the constraints and variables nature and of the parameters values. Hence the structural model of a system represents its structural properties, i.e. those properties which are independent of the nature of the system variables and their relations and of the parameters values.

Structural analysis then consists in finding out the system structural properties, i.e. it resorts in the study of the structural graph of the system. For this reason first a definition of the structural graph is stated:

Definition 3.1. Structural Graph.
The structural graph of a system is a bi-partite graph with two sub-sets of vertices: V set of the system variables, and C set of the system (static or dynamic) constraints. The set of edges E is defined by:

$$(v, c) \in E \subseteq V \times C \iff \text{the variable } v \text{ is an argument of the constraint } c. \tag{3.1}$$

To any constraint $c \in C$ is associated the set $V(c)$ of all the variables such that

$$\omega \in V(c) \iff (\omega, c) \in E \tag{3.2}$$

In the same way to each variable $v \in V$ is associated the set $C(v)$ of all the constraints such that:

$$\gamma \in C(v) \iff (v, \gamma) \in E. \tag{3.3}$$

Structural graphs are non oriented by definition, but matchings are used to provide an orientation.

Definition 3.2. Matching.
A matching M on a bi-partite graph is a subset of E such that

$$\forall (v,c), (v',c') \in M : v \neq v' \wedge c \neq c'. \tag{3.4}$$

Given a matching M, different cases can be distinguished:

1. $|M| = \min\{|V|, |F|\}$: the matching is **complete** with respect to the variables ($|V| < |F|$) or to the constraints ($|V| > |F|$) or to both of them ($|V| = |F|$). A complete matching is **maximal**, which means that no matching of larger cardinality exists.

2. $|M| < \min\{|V|, |F|\}$: the matching is not complete, it may be maximal or not.

Let (v,c) be an edge belonging to a matching: v is called the **output** of constraint c: $v \triangleq o(c)$, while $V(c)\backslash\{v\}$ is the set of its **inputs**: $V(c)\backslash\{v\} \triangleq I(c)$.
Let $\{c_1, c_2, ...c_q\}$ be a subset of constraints (without repetition) such that

$$i = 1, 2, ...q-1 : o(c_i) \in I(c_{i+1}) \tag{3.5}$$

Definition 3.3. Alternated Chain.
The path $c_1 - o(c_1) - c_2 - o(c_2) - ... - c_q$ is an alternated chain between constraint c_1 and constraint c_q.

Each oriented graph representing a matching has the following property: every existing path between two variables is an alternated chain.

Definition 3.4. A **loop** is a subset of constraints $\{c_1, c_2, ...c_q\}$ such that for any pair (c_i, c_j) there is an alternated chain from c_i to c_j and another one from c_j to c_i.

There are many methods to eliminate loops from bipartite graphs, such as graphs condensation. For a complete presentation of these procedure the interested reader can refer to [23]. It will be seen in next subsection that it is extremely important to eliminate loops in structural graphs representing system functions if we are interested in the detection of losses of functions.

As stated at the beginning of this subsection, structural analysis aims at the study of structural properties of a system, and this is obtained using the structural graph. Structural properties of interest in fault detection and reconfiguration are

- *monitorability (or diagnosability)*: identification of the subset of the system components in which faults can be detected and isolated,

- *analytical redundancy*: this property is related to the previous one because it leads to the possibility to design residual signals (for fault diagnosis) which are robust (i.e. insensitive to disturbances and uncertainties) and structured (i.e. sensitive to certain faults and insensitive to others),

- *reconfigurability*: capability of finding alternative paths to the variables of interest in case of a fault (sensor, actuator or system component failures), in order to still be able to estimate/control them.

Before stating the theory to find out if a structural graph has the above mentioned properties, it is necessary to give some definitions on the standard structural properties. From def. 3.2 and 3.3, the following notion of reachability is obtained:

Definition 3.5. Reachability.
A variable v_2 is reachable from a variable v_1 iff there exists an alternated chain from v_1 to v_2. A variable v_2 is reachable from a subset $\chi \subseteq V \setminus \{v_2\}$ iff there exists $v_1 \in \chi$ such that v_2 is reachable from v_1. A subset of variables V_2 is reachable from a subset of variables V_1 iff any variable of V_2 is reachable from V_1.

Recalling the bi-partite graph theory, it is possible to divide the structural graph of a system $S = (C, V)$ in three subgraphs representing three subsystems:

- $S^+ = (C^+, V^+)$ is called the over-constrained system,
- $S^0 = (C^0, V^+ \cup V^0)$ is called the just-constrained systems,
- $S^- = (C^-, V^+ \cup V^0 \cup V^-)$ is called the under-constrained system;

where

- (C^-, C^0, C^+) are a partition of C,
- (V^-, V^0, V^+) are a partition of V,
- (C^+, V^+) is a over-constrained graph, i.e. there exists a complete matching on the variables V^+ but not on the constraints C^+,
- (C^0, V^0) is a just-constrained graph, i.e. there exists a complete matching on the variables V^0 and on the constraints C^0,
- (C^-, V^-) is a under-constrained graph, i.e. there exists a complete matching on the constraints C^- but not on the variables V^-.

The over-constrained (resp. the just-constrained) subsystem is called *causal* iff all the variables V^+ (resp. V^0) can be matched without introducing any differential loop, i.e. a loop of the kind:

$$\begin{aligned} v_l &= g_l(x_l, x_e, u) \\ v_l &= \tfrac{d}{dt} x_l \end{aligned}$$

where the subscript l defines the items in the loop and the subscript e defines the external (known) items.

System variables V are decomposed into known and unknown ones: $V = K \cup X$. Known variables K can be sensor outputs, reference inputs, controllers and observers outputs. Observability of a system is related to the possibility of computing the values of the unknown variables using the known ones. Then if the set of constraints C is decomposed in:

$$\begin{aligned} C_K &= \{c \in C : V(c) \cap X = \emptyset\} \\ C_X &= \{c \in C : V(c) \cap X \neq \emptyset\} \end{aligned}$$

the analysis of observability refers only to the subgraph $< C_X, X, E_X >$ with canonical decomposition given by:

$$\begin{aligned} S^+ &= (C_X^+, X^+) \\ S^0 &= (C_X^0, X^+ \cup X^0) \\ S^- &= (C_X^-, X^+ \cup X^0 \cup X^-). \end{aligned} \tag{3.6}$$

The following theorem can then be stated:

Theorem 3.1. Structural Observability.
A necessary and sufficient condition for a system with canonical decomposition 3.6 to be structurally observable is that, under derivative causality:

- all the unknown variables are reachable from the known ones,
- the over-constrained and the just-constrained subsystems are causal,
- the under-constrained subsystem is empty.

In the structural graph this is represented through **estimation subgraphs**, i.e. sets of alternated chains relating the target variable (for which observability is inquired) to the set of the known ones, meaning that no non-observable variable acts as input in any constraint of this graph.

From the notion of observability it comes the property of monitorability (or diagnosability). A system is **monitorable** if it can be determined (starting from the known variables) even when the constraints are not all satisfied. Monitorability is strictly related to the possibility of designing **Analytical Redundancy Relations** (ARRs) starting from the constraints of the system, i.e. if there are some constraints relating the time evolution of known variables in nominal operating conditions. The fault detection procedure checks if the ARRs are satisfied or not, if not then a fault has occurred. Indeed in structural analysis a fault is defined as a change in some constraints. From that the following definition is derived:

Definition 3.6. Structurally Monitorable Subsystem.
The structurally monitorable part of a system is the subset of constraints for which there exists ARRs structurally sensitive to their changes.

A theorem to detect if a fault is monitorable is then:

Theorem 3.2. Monitorability.
Two equivalent necessary conditions for a fault φ to be monitorable are:

1. $X_\varphi = V(\varphi) \cap X$ is structurally observable (i.e. $X_\varphi \subseteq X_{obs}$) in the system $(C \setminus \{\varphi\}, V)$,
2. φ belongs to the structurally observable over-constrained part of the system (C, V).

Depending on the considered system, there are different definition of **controllability**, i.e. the possibility of finding control (unknown input variables) able to let the system variables achieve some desired values.

An interesting theorem can be derived for a system of the kind:

$$\begin{aligned} \dot{x} &= f(x, u, t) \\ \dot{x} &= \tfrac{d}{dt}x \end{aligned}$$

where f is linear, $\dot{x}$ is known, $x,\ u$ are unknown.

Theorem 3.3. Structural Controllability for Linear Continuous Systems.
Necessary and sufficient conditions for the above system to be structurally controllable is that:

- K is reachable from the inputs,
- the canonical decomposition of $< C_X, X, E_X >$ contains no over-constrained subsystem.

In the structural graphs controllability is analysed using **control subgraphs**, i.e. sets of alternated chains relating the target variable (for which controllability is enquired) and the set of the inputs.

From a fault tolerant point of view, these properties can be used to decide the strategy to follow to reconfigure a system after a fault. In case the fault does not change the structure of the systems, i.e. the change its occurrence provokes to a constraint can be detected, isolate and estimated (which means that the system is still known), the observability and controllability properties remain the same. This strategy is called **fault accommodatio**n and is based on the design of "good" residual signals.

If the fault changes the structure of the system, i.e. the changes in the constraints can only be detected and isolated, the structural properties of the system obviously change. The problem is now to decide if the system is reconfigurable. In other words, the analysis of the reconfigurability of a the control scheme resides in the decision on the observability and/or controllability of a target variable if some components are lost, i.e. some vertices and the related edges are removed from the structural graph. The target variable is observable (controllable) if at least one estimation (control) subgraph with the same target is still in the graph when these vertices are removed. This reminds to the computation of these subgraphs, meaning to the computation of redundancy degrees in the system.

Definition 3.7. Redundancy Degree.
The estimation (control) redundancy degree of an observable variable x w.r.t. a fault φ is the number of estimation (control) subgraphs with target x not affected by φ.

The redundancy degree can be used as a measure of **reconfigurability** of the estimation or control problem.

A deeper insight on structural analysis is presented in [5] and references therein, while the application of structural analysis to FDI and FTC is studied in [67], [31] and [32].

3.1.3 From Structural Analysis to Functional Representation

In the following an algorithm to cast structural graphs in functional trees is introduced. The concept has been presented in [16].

Starting from the objective structural graph introduced in previous section, the functional representation of a system can be obtained in a procedural way. Indeed a connection between the functional and the structural representation of the system can be stated. This basically relies on the assumption that system components have been selected or designed so as to fulfill some given functional goals, which can be translated into properties that some system variables must satisfy. Indeed the system performs a given function ϕ_n by providing one of its variables, say v, with a given property $P_n(v)$ (e.g. v is wanted to track a specific reference trajectory). Therefore, it is advisable to exploit of the structural model to automatically derive functional trees from the system structural graph.

In this subsection with the word system both the whole distributed system and its nodes (subsystems) will be referred, while the strategy to design the functional trees and the structural graphs is the same. Nevertheless it has to be pointed out that for each structural graph there can be many different main functions and each of them can be considered as a root of a tree.

With the same policy used to build estimation and control graphs (see previous subsection), it is necessary to choose a target variable representing the main function the system has to achieve. This variable, or better the property associated to it, will be the root of the functional tree. Always using the same idea of the analysis of structural properties (reachability, observability,

etc.), some variables in the graph will be known because measured. Generally these variables are the inputs and outputs of the system, meaning variables associated with sensors outputs and control variables, i.e. outputs of the control laws. These considerations are the basis to design the functional tree in order to decide the root of the tree and to find a path in the structural graph from the known variables to this target. The known variables will then be the leaves of the tree, i.e. the ground functions.

Using the notation introduced in previous subsection, consider the set $C(v)$ of all the constraints in which v is present. Then v could be matched in any of these constraints:

$$v = \vee_{c \in C(v)} o(c). \tag{3.7}$$

Now, if v is matched in constraint $c \in C(v)$, then for v to exhibit the property $P_n(v)$ it is necessary that the subset of variables $I(c)$, which are inputs of constraint c, exhibit some (other) property, say P_{n-1}. This can be expressed as

$$\forall w \in I(c) : P_n(v) \Rightarrow P_{n-1}(w). \tag{3.8}$$

From Def. 3.2 and eq. (3.8) it comes straightforward that the orientation induced by the matching has a **causal interpretation** when associating functional properties to the variables. Indeed causality is given by the necessity of some properties to be satisfied for the satisfaction of other properties in matched variables. This causal behaviour gives exactly the relations among different functions represented by the functional tree. For this reason a tree having variable v, i.e. its property $P_n(v)$, as root has then the following properties:

- v is connected with all constraints $c \in C(v)$ by an OR node: property $P_n(v)$ can be obtained by a causal link associated with any matching of the set described by (3.7),
- each constraint $c \in C(v)$ is connected to its input variables $I(c)$ (if more than one) by an AND node: the output $v = o(c)$ can have property P_n only if all the variables $I(c)$ exhibit property P_{n-1}.

To write an algorithm starting from the above stated considerations it is first necessary to choose in the structural graph a root variable representing the main property the system must achieve. Then to detect all the measure and control variables and find all possible matchings in which these variables are not matched (leaves of the tree).

Starting from the root variable the algorithm can be written in pseudo-code as in Algorithm 1.

Algorithm 1 From Structural Graph to Functional Tree

```
while There are constraints available for matching do
  for all variables z at level k do
    associate an OR node which concentrates all constraints available for matching in C(z)
    for all constraints γ ∈ C(z) do
      associate an AND node which concentrates all variables V(γ)\z.
    end for
  end for
end while
Level k + 1 is built on all variables that are concentrated by AND nodes.
```

The procedure ends when the non-matched variables are reached. Note that constraints available for matching are all constraints except those already used at higher levels.

The functional tree designed in this way has the same properties of the structural graph. In particular the analysis of the structural graph can lead to the same results in the functional trees. This can be exploited for fault detection and reconfiguration purposes. Detectable and reconfigurable functions in the fault tree are related to monitorablity and reconfigurability of the corresponding variables. Using the definitions and theorems stated in the previous subsection, diagnosable and/or reconfigurable variables are found in the structural graph, then the properties on these variables, namely the corresponding functions in the obtained tree, are diagnosable (meaning that the loss of the function associate of a diagnosable variable is detectable) and/or reconfigurable (meaning that if the function is lost, it is possible to use the other functions to achieve again the main function or a degraded version of it). In Sec. 3.3 it will be shown how to use these considerations to decompose the fault detection and reconfiguration tasks following a functional decomposition.

The following example explains how the above introduced procedure works.

Example 3.1. *The tank system*
Consider the tank system represented in fig. 3.3. The system is controlled via an actuator acting on the throttling of the valve on the pipe providing the input flow. It has also a redundant actuator (with the corresponding valve and pipe) which is supposed not to act in nominal conditions but it can be used to reconfigure the system in case the nominal actuator fails. The system is also controlled by an actuator acting on the valve of the output pipe.

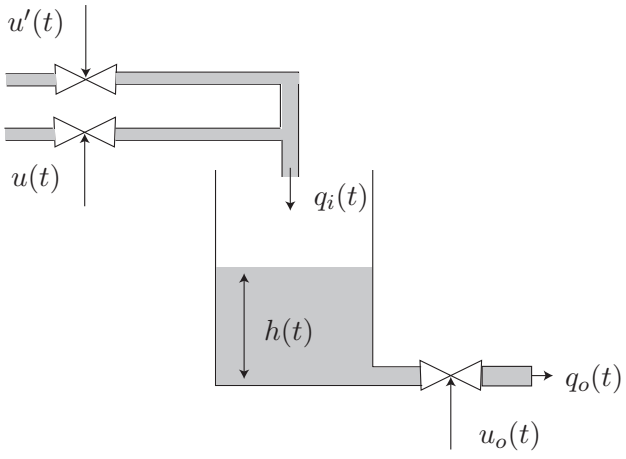


Figure 3.3: The tank system.

Its behaviour model is given by:

$$
\begin{array}{ll}
\text{Tank } c_1: & \dot{h}(t) = q_i(t) - q_o(t) \\
\text{Input valve } c_2: & q_i(t) = \alpha u(t) \\
\text{Input valve } c_{2'}: & q_i(t) = \alpha u'(t) \\
\text{Output pipe } c_3: & q_o(t) = k(t)\sqrt{h(t)} \\
\text{Level sensor } c_4: & y(t) = h(t) \\
\text{Control algorithm } c_5: & u(t) = f(y) \\
\text{Control algorithm } c_{5'}: & u'(t) = f(y) \\
\text{Derivative constraint } c_6: & \dot{h}(t) = \dfrac{dh(t)}{dt} \\
\text{Output valve } c_7: & k(t) = \beta u_o(t) \\
\text{Output control } c_8: & u_o(t) = f_o(y)
\end{array}
\tag{3.9}
$$

where u, u' and u_o are the actuators outputs, h denotes the liquid level, q_i and q_o the flow into or out of the tank, while α and β are the valves constants. Each component introduces one constraint. The extra constraint c_6 expresses the fact that $\dot{h}(t)$ is the derivative of the level $h(t)$; we suppose to know initial conditions, so it is always possible to know $\dot{h}(t)$ from $h(t)$ and viceversa.

The structural graph of the tank system described by model (3.9) is depicted in fig. 3.4.

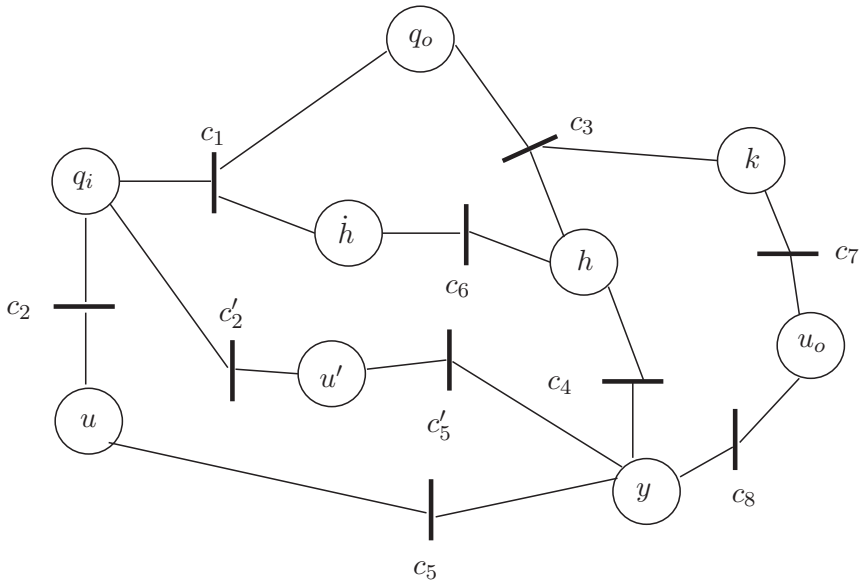


Figure 3.4: Structural graph of the tank system.

After graph condensation (see [23]) to eliminate the physical loop $\dot{h}$, h, q_o it is possible to build some possible matchings for system (3.9). They are reported in tables 3.1, 3.2 and 3.3. Note that some constraints are not matched, and therefore the matchings are neither maximal nor complete with respect to the variables or the constraints.

The above introduced procedure to design a functional tree starting from the structural graph of the system is now applied to system (3.9). The main functionality of the system is to make level h reach a certain value h_0. Variable h is chosen as root but due to the loop condensation

Table 3.1: A possible matching (I) for (3.9).

	$(\dot{h},\, h,\, q_o)$	y	u	u'	q_i	u_o	k
$(c_1,\, c_3,\, c_6)$	①				1		1
c_2			1		①		
c_2'				1	1		
c_4	1	1					
c_5		1	1				
c_5'		1		1			
c_7						1	①
c_8		1				1	

Table 3.2: A possible matching (II) for (3.9).

	$(\dot{h},\, h,\, q_o)$	y	u	u'	q_i	u_o	k
$(c_1,\, c_3,\, c_6)$	①				1		1
c_2			1		1		
c_2'				1	①		
c_4	1	1					
c_5		1	1				
c_5'		1		1			
c_7						1	①
c_8		1				1	

Table 3.3: A possible matching (III) for (3.9).

	$(\dot{h},\, h,\, q_o)$	y	u	u'	q_i	u_o	k
$(c_1,\, c_3,\, c_6)$	1				1		1
c_2			1		①		
c_2'				1	1		
c_4	①	1					
c_5		1	1				
c_5'		1		1			
c_7						1	①
c_8		1				1	

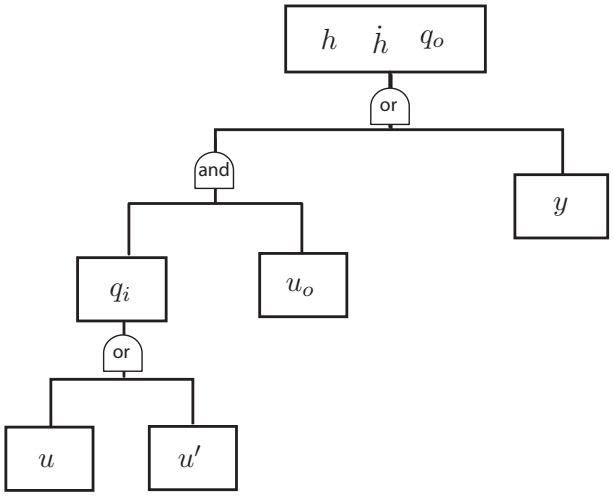


Figure 3.5: The tank functional trees.

knowing the level h is equivalent to the knowledge of its derivative and to the knowledge of the output flow q_o, this means that the functionality will be the same. Starting from the matchings in tables 3.1, 3.2 and 3.3 and following the procedure algorithm, the tree represented in fig. 3.5 is obtained. It is possible to see that the two redundant actuator variables are related to the upper level variable with an OR gate. Moreover an analytical redundancy is presented here, because the knowledge of the main functionality can be achieved through the measure of the level. But both the knowledge of the inflow and of the output actuator are needed, which is the meaning of the AND gate.

3.2 A Modular Hierarchical Architecture for Distributed Systems

The three-layers modular architecture for fault tolerant control of distributed systems developed in the framework of the European Project *IFATIS* (Intelligent Fault Tolerant Control in Integrated Systems, IST-2001-32122) is sketched in fig.3.6 (see ... for a complete description).

Following the distributed nature of the considered system the architecture relies on a first decomposition of the system itself based on a functional criterion (as presented in subsection 3.1.1). The system is then divided into partial processes, representing the different nodes of the distributed system (see Sec....).

The main function associated to each partial process can be:

- *a control function*: if the main objective of the partial process is control.
- *a measurement function*: if the main objective of the partial process is measure.

The partial processes in their turn are represented using a two-layers architecture: a physical level, the *plant*, and a control/supervisor level, called *Fault Tolerant Control or Measurement (FTC/M) Module*. The FTC/M modules, as the name says, have the aim of achieving control (measure) of the plant in a fault tolerant way, i.e. producing some diagnostic signals and reconfiguration policies to recover the task of the process in case of fault. A set of operational modes can be associated to each partial process, some of them are nominal, some other represent

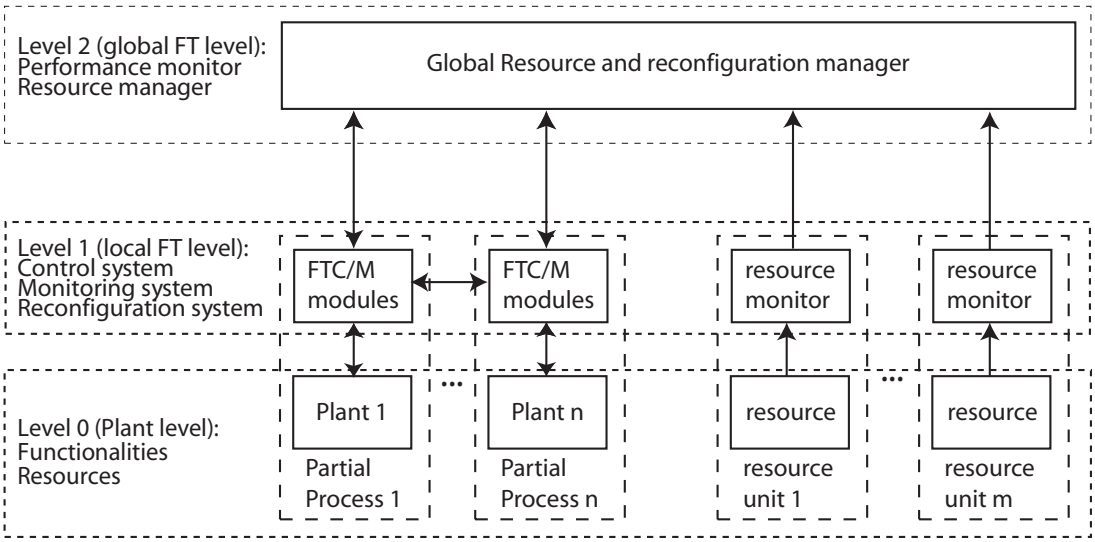


Figure 3.6: IFATIS architecture.

reconfiguration possibilities after a fault occurrence. These modes are called **Working Modes (WM)** and are decided by the FTC/M modules.

The *resource units* laying at the lower level in fig. 3.6 represent physical/logical resources needing dedicated diagnosis. These resources are particularly important for the system and their loss can lead to the loss of a basic function for the achievement of the main task. For this reason they have dedicated diagnostics and generally they are duplicated to obtain hardware redundancy.

The higher level of the architecture is represented by the *Global Resource and Reconfiguration Manager* (GRRM). Its aim is to generally supervise the whole system. Having a global knowledge of the system, this supervisor can validate the WM changing requests given by the lower levels, if these involve more than one partial process reconfiguration, resources reallocation or contrasts in different partial process supervisors decisions. The GRRM also manages reconfiguration for resource faults and is an interface with the external world, i.e. with the human operator. In other words, the requests of WM changes raised by the local modules must be elaborated at system level, taking into account global resources and global objectives, before really actuating the control reconfiguration. This task is done at the higher level of the architecture by the GRRM.

In a general framework the tasks demanded from a supervisor are represented in fig. 3.7. The *Fault Detection and Isolation (FDI) unit* provides, on the basis of the different diagnostic (residual) signals, an estimation of the failure mode observed in a specific process/sub-process. The failure mode estimation is then processed by the *Event Generator (EG) unit* whose aim is to raise requests of working mode changes to react to the presence of determined faults. Finally the requests of WM changes must be managed by a *decision logic unit (DLU)* able to validate the WM changes requests according to specifications on the part of the system it supervises.

In the above description of the different levels of the supervisory architecture we find this kind of supervisor in each FTC/M module and in the GRRM, with the only difference that the GRRM has no diagnostic part because all the diagnostics are done at the lower levels. This is due to the fact that the diagnosis involves only the failure modes that can be detected and isolated at the lower levels, without need of a global process vision, given a well posed functional

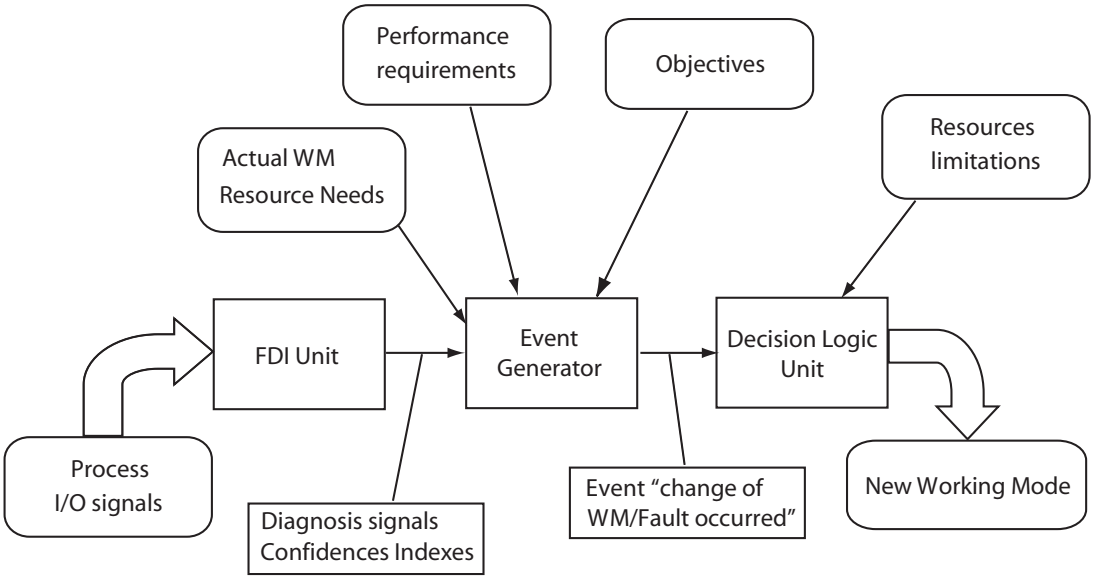


Figure 3.7: Supervisor structure.

decomposition of the system. Nevertheless all the diagnostic signals are sent to the GRRM to be processed by its supervisor to give the final decision on the reconfiguration policy.

The FDI unit embeds the diagnostic task, i.e. it is the part of the supervisor able to give a response on the possible fault occurrence. From a functional point of view the FDI unit does not give an estimate of the physical fault, but it produces an estimate of the lost function (failure mode). The output of the FDI unit is then a signal embedding the estimate of the failure mode and possibly a reliability index (for reliability analysis the reader is referred to [4] for a general framework and [7] for the application to distributed systems).

The output of the FDI unit is then processed by the EG unit. Its task is to validate the signal raised by the FDI unit. This task is accomplished using the information contained in specifications on performance requirements, resource needs and process objectives. Moreover the EG unit receives all the signals raised by the different FDI unit of the process and has to decide which is the more reliable to be occurred. The EG, knowing the actual WM of the system, decides the reconfiguration strategy. The decision is taken according to some "optimal"policy (see as an example [33] and [34]). In its simplest version the EG unit can be a look-up table designed to respect the above mentioned specifications.

The output of the EG is an activating signal, i.e. it is the signal which will be used by the DLU to activate the next working mode. The DLU represents the possibilities of reconfiguration for a particular function/process. This means that all the possible successive working modes after the loss of a function are embedded in the DLU. In this sense the DL can be modeled using the Discrete Event Systems (DES) methods. Using the supervisory theory of DES (see [18]) the DLU of a certain function/process is the automaton modeling the specifications for supervising the function/process. This means that a DES model of the function/process itself is needed. The design of the decision logic unit is then performed in two steps: first the function/process is modeled as a DES, then the supervisory specifications for this function/process are designed.

The different roles and tasks of the supervisor will be better explained in next section, where a method to obtain a decentralisation of the supervisor tasks is introduced.

3.3 Distributed Diagnosis and Supervision

Following the same functional criterion used to build the architecture presented in the previous section, it is possible to find a hierarchical decomposition in each FTC/M module. This decomposition is motivated by the necessity of supervising very complex systems, for which nodes are still complex systems.

A further decomposition in sub-functions inside each node lead to a modular structure for diagnosis and reconfiguration. In this section the procedure to achieve fault detection and reconfiguration throughout the architecture is presented, replicating the different tasks (units in fig. 3.7) of the supervisor at different levels. First the diagnostic task is performed at lower level, using the fault tree to decompose the system following the losses of functions, and the structural policies to establish the detectable functions, i.e. failure modes. At this first level the diagnostic signals are produced and their interpretation is given. The second step of the procedure gives the reconfiguration suggestions. At lower level the functional tree is used to decompose the system in functions and, always using structural considerations, the reconfigurable functions are detected. The reconfiguration policies suggested at lower level must be validated by the higher level supervisor. The real logic behind the working modes switching is realised using a discrete event representation.

The procedure introduced in this section is based on the work presented in [6].

3.3.1 Lower Level Diagnosis

Each partial process represents a main function of the system, i.e. it represents a node of the distributed system. This means that for each partial process it is possible to build a functional tree and its associated fault tree. Fault trees, as described in Sec. 3.1, introduce the new concept of failure, i.e. of effect of the fault in the functions of the system. While each node represents a loss of a function, the fault tree of a system is a tool for detection of the losses of functions at different levels. Having defined *failure modes* as the *detectable losses of functionalities*, the procedure presented in Subsection 3.1.3 is useful to establish the failure modes in the fault tree. After their generation with the help of structural considerations, the residual signals can be associated to the failure modes they are able to detect.

In a graphical representation, some residual signals are related to the nodes of the fault trees representing the failure modes detectable using them. This step leads to the identification of a hierarchy in the residual evaluation algorithm of the FDI unit. A nested interpretation can be given to the overall residual matrix, i.e. the residual matrix can be obtained by nesting different residual matrices related to different levels of the fault tree. Using the residual matrices it is possible to perform the detection of a loss of function and the isolation of the failure mode closer to the physical faults (graphically represented by the leaves of the tree). There are two possible directions of interpretation of the residual matrix obtained in this way. Following a bottom-up analysis of the tree with the associated residual signals and matrices, the identification of the failure modes can be performed. The physical fault identification is performed via a top-down analysis, using the residual matrices associated to each failure mode, i.e. a deeper analysis of the residual matrices associated to each failure mode can be used to specify to which physical fault(s) the failure is due. This information can be used by the EG to choose the optimal reconfiguration policy if based on the combinations of faults (see next subsection). Using this method it is possible to decentralise the diagnosis for complex systems, composed by a big number of functions.

As stated in Sec. 3.2 in fig. 3.7 detection and isolation is performed by the FDI unit. With the above presented procedure each detectable loss of function introduces an associated FDI

unit. Referring to fig. 3.6 the signal produced by the FDI unit is the estimation of the occurred failure mode, or a list of possibly occurred failure modes with associated reliability measures.

The following example can be useful to better understand the above procedure and its advantages for distributed fault detection and isolation.

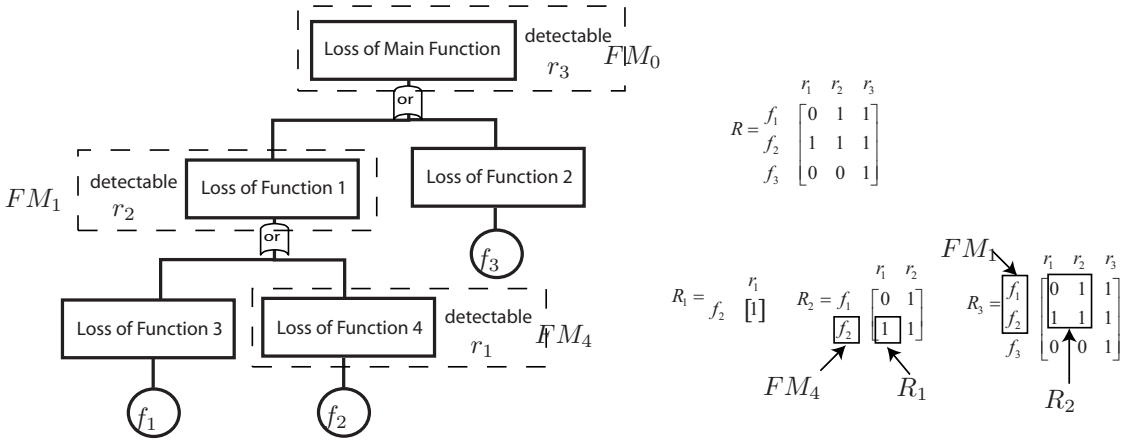


Figure 3.8: Example 3.2: fault tree and nested residual matrices.

Example 3.2. Consider the fault tree in fig. 3.8. Suppose it represents the fault tree of a partial process in a wider distributed system. Suppose also that using the procedure explained in Subsection 3.1.3 the only three detectable losses of function are the main one, function 1 and function 4. Their losses are detected using residuals r_1, r_2, r_3. In fig. 3.8 each residual in associated to the failure mode it is supposed to detect. The overall residual matrix R can be thought as given by the composition of three sub-matrices related to the three failure modes. In fact matrix R_1 is able to detect failure mode FM_4. Then matrix R_2, able to detect failure mode FM_1, expresses the relationship between the residual signals available at that level (r_1 and r_2) and the failure modes below FM_1. It is possible to see how matrix R_1 is nested in R_2. The final residual matrix $R_3 = R$, related to the failure mode FM_0 and yielding the relationship between all the residual signals at the top level and all the failure modes below, embeds the matrix R_2. In this respect the whole residual matrix can be designed through a bottom-up procedure by inspection of the fault tree.

To explain how this interpretation can be used supposed that FM_1 is occurred. This means that r_2 is raised. Then the FDI unit associated to FM_1 sends a signal to the EG to say that the function has failed. Now if the EG needs to know to which physical fault between f_1 and f_2 the failure is due, the analysis of the residual matrix R_2 of FM_1 must be performed, looking at the nested residual matrix R_1 of FM_4. If R_1 says that even FM_4 has occurred then the failure of function 1 is due to fault f_2.

This example is obviously very simple, because in the case above described the fault f_2 will lead both to the loss of FM_1 and FM_4 so both FDI units of the respective functions will send a signal. Moreover in this example multiple faults are not considered. Nevertheless it is easy to extend this policy to more complex systems, decentralising the first decision on the failure mode occurred.

3.3.2 Lower Level Supervision

The diagnostic results produced by the FDI units at lower level are then processed by the supervisory units at the same FTC/M level. Even for supervision it is possible to find a hierarchy inside each node. To this end the graphical description given by the functional tree can be used. Define *Reconfigurable Functions* those functions in the functional tree on which the designer has some degree of controllability. The decomposition is based on the capability of finding the reconfigurable functions starting from the knowledge of the system in a procedural way.

Nevertheless identification of reconfigurable functions is, in general, strongly application dependent. As presented in Sec. 3.1.3 it is possible to use structural analysis to identify over-constrained controlled variables and relate these results to the functional tree.

For each reconfigurable function a set of possible working modes is established. In this framework they aim at reconfiguring the lost function to achieve again the same or degraded performances, depending on the severity of the failure. The working modes then represent different behaviours of the whole system, obtained reconfiguring the lost function with some methods (see [5] for an excursus on reconfiguration techniques).

Following the same idea of the diagnostic procedure, it is possible to associate a set of WMs to each reconfigurable function in the functional tree. This is the set of all the possible reconfiguration actions in case the function is lost. As done for detection, each reconfigurable loss of function introduces a decision logic unit. Inside this unit a logic graph representing all the possibilities of reconfigurations activates the working mode decided by the event generator.

While the EG takes the decision on the policy to follow after a failure of one of the functions of the process, each partial process as a whole has its own event generator. The event generator receives the information on the possibly occurred failure and decides the counteraction following some optimal policy. This means that the EG receives all the signals from the FDI units of the partial process and decides which failure has occurred. The signal produced by the event generator is then input of the decision logic unit. The event generator task cannot be distributed because it needs the complete knowledge of the process to take an optimal decision, which indeed depends on the collective behaviour of the different functions composing the partial process itself. The event generator sends down to the different levels of the partial process the reconfiguration signal decided with its optimal policy. This signal activates one of the possibilities of one of the decision logic units. This decision has to be confirmed by the higher level supervisor embedded in the GRRM, then the signal produced by the EG is also sent to the GRRM.

The right decision logic unit receives the signal from the EG which activates one of the reconfiguration possibilities. To this end the DLU is modelled using a state machine, a tool able to represent the switching possibilities between the different working modes and the external signals. Using the theory of discrete event systems (see [18]) the automaton inside each DLU gives the specification description for the process to be supervised. Introducing some notation, define:

- $WM0_j^i$ the nominal working mode(s) state of the j-th reconfigurable function in the i-th partial process;
- WMk_j^i the k-th reconfigured working mode state of the j-th reconfigurable function in the i-th partial process;
- F_j^i the faulty state representing the loss of j-th reconfigurable function in the i-th partial process;

- wmk_j^i the controllable event by which the supervisor reconfigure the j-th function in the i-th partial process by forcing the k-th working mode;
- $wm0_j^i$ the controllable event by which the supervisor resets the j-th function in the i-th partial process at its nominal working mode;
- f_j^i is the non-controllable but observable (because of diagnosis) event occurring in case the j-th reconfigurable function in the $i-th$ partial process is lost due to the occurrence of a failure;
- a_{jk}^i is the *controllable* event used by the EG to activate the k-th WM in the supervisor of the j-th reconfigurable function in the i-th partial process;
- G_j^i denotes the automaton modelling the behaviour of the j-th reconfigurable function in the i-th partial process;
- H_j^i denotes the automaton modeling the reconfiguration specifications behind the j-th reconfigurable functionality in the i-th partial process;
- K_j^i denotes the automaton modeling the controlled behavior linked to the j-th reconfigurable functionality in the i-th partial process.

The design of the DLU basically comprises in the design of the automaton representing the specifications for each reconfigurable function.

The automaton G_j^i of each function j of the partial process i is designed following some basic guidelines: starting from the nominal WM(s) $WM0_j^i$ the occurrence of the failure event f_j^i activates the faulty state F_j^i; the events a_{jk}^i leave the automaton in state F_j^i; through the events wmk_j^i the automaton moves then to the reconfigured states WMk_j^i; finally using the events $wm0_j^i$ the system is moved back to the nominal state(s).

The automata representing the DLUs are then designed starting from G_j^i. The states lose their meaning, because the specifications are given on the strings of events: after event f_j^i, the activating event a_{jk}^i allows the event wmk_j^i to occur. After that the partial process will be in working mode WMk_j^i.

This way of designing the specifications in the DLU considers the possibility of a failure to be due to different combinations of physical faults, so that different reconfigurations are possible. The EG then decides which combination is more likely occurred and raises the corresponding activating signal (as explained in the previous subsection with the use of residual matrices).

The procedure is again better clarified by the following example.

Example 3.3. Referring to example 3.2, in fig. 3.9 the functional tree referring to the fault tree in fig. 3.8 is reported. The reconfigurable functions are in this case function 3, function 1 and the main function. To each reconfigurable function a set of WMs is associated. Now suppose that to the reconfigurable function 1 three working modes are associated: $WM1_1^1$, $WM2_1^1$ and $WM3_1^1$. Then the automaton describing the behaviour of function 1 is represented in fig. 3.10(a) where $WM0_1^1$ is the nominal working mode and $a_{1h}^1, h \in 1,2,3$ are the activating events for the three working modes. Note that for the sake of simplicity here the resetting event $wm0_1^1$ has been omitted. The specifications for the DL are then: if a_{1k}^1 is active then working mode WMk_1^1 must be forced. These specifications are represented by the automaton of fig. 3.10(b). One possible choice for the event generator of the process in this case is to check what combination of faults is occurred: if fault f_1 is occurred then the EG raises event a_{11}^1, if fault f_2 is occurred then the

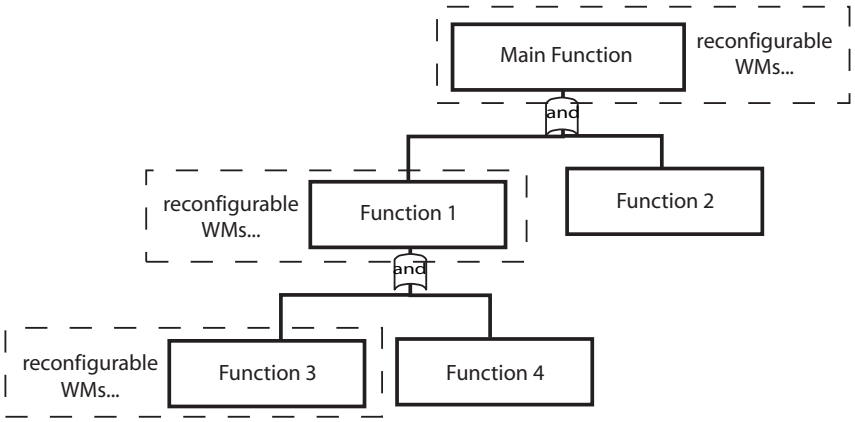


Figure 3.9: Example 3.3: functional tree and reconfigurable functions.

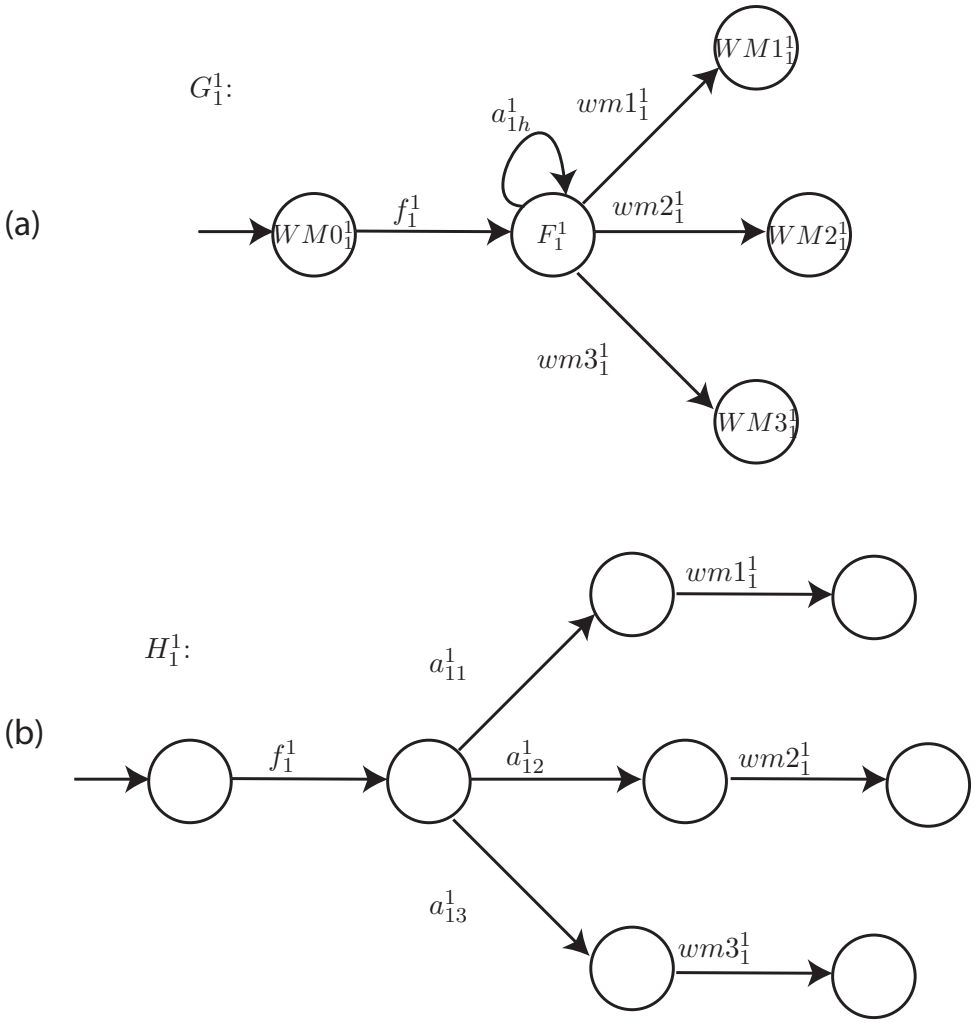


Figure 3.10: Example 3.3: (a) behaviour of function 1; (b) specifications for function 1.

EG raises event a^1_{12}, if both faults f_1 and f_2 are occurred then the EG raises event a^1_{13}. The EG can use the strategy presented in Example 3.2 to decide what fault has occurred.

It is worth noting that the signals for failure detection are sent from the bottom of the architecture to the higher level, whereas the signals for function reconfiguration go from the top of the architecture down to the different layers of the system. This follows the standard principles of fault tolerance, because fault detection has to be performed first and can be achieved even at local level, while the decision on the reconfiguration strategy needs first an estimate of the occurred failure and then a complete knowledge of the propagation of the failure effect on the other parts of the system. For this reason the reconfiguration strategy here proposed is distributed in the way the reconfigurable functions are detected and introduces a tool to represent the possible reconfigured working modes at local level, but the final decision has always to be delegated to a supervisor with a broader vision of the process.

3.3.3 Higher Level Supervision

At higher level the reconfiguration part of the supervisor (event generator and decision logic) is replied. The GRRM, as described in Sec. 3.2, aims basically at validating the WM request changes issued by the local supervisors and at managing the resource faults and reallocation.

To this end the event generator in the GRRM has to take into account some general requirements for the decision logic unit.

Processes allocation. Off-line planning on how the working modes of the different processes can be allocated on the available resources, depending on the resource needs of the WMs.

Working Modes overlapping. While the GRRM has the possibility to check the consistency of the working modes raised by local supervisors, it can inhibit the request of working mode change in case of conflict.

Working Modes Hierarchy. In case of limited resources (due to resources faults or heavy reconfiguration policies) a hierarchy among the WMs of the different sub-processes has to be established. This hierarchy depends on the urgency of the reconfiguration and it is only known by the GRRM because of its global view.

External signals. Some reconfiguration actions can be imposed by external (human operator) requirement represented by signals sent to the GRRM which is the interface with the external world.

The GRRM needs also to interface with the lower level logic. For this reason the decision logic is always represented as an automaton giving the specification for the supervision of the system.

The first step is then the design of the automaton representing the whole system. This automaton will include all the automata of the partial processes and the automata of the resources.

To build the automaton G^i of each partial process i, the automata G^i_j of each function j are composed using parallel composition. The same applies for the automaton H^i give by the parallel composition of all the automata $H^i_j, \forall j$. Finally automaton K^i representing the controlled behaviour of the i-th partial process is obtained as parallel composition of G^i and H^i.

The automata of the resources are designed using the following notation:

- I_ℓ represents the idle condition state for the ℓ-th physical resource;
- B_ℓ represents the busy condition state for the ℓ-th physical resource;

- SB_ℓ represents the standby state for the ℓ-th physical resource;
- F_ℓ represents the faulty state for the ℓ-th physical resource;
- SB_ℓ^i represents the standby state for the i-th partial process over the ℓ-th physical resource;
- sb_ℓ is the controllable and observable event used by the GRRM to force in standby the ℓ-th physical resource;
- on_ℓ is the controllable and observable event used by the GRRM to turn on the ℓ-th physical resource;
- f_ℓ is the non-controllable but observable resource fault event arising whenever the local resource monitor detects a fault in the ℓ-th physical resource;
- R_ℓ denotes the automaton modelling the uncontrolled behavior of the ℓ-th physical resource.

The automaton of a resource is generally designed in this way: the resource is nominally in standby state SB_ℓ; the turn-on event on_ℓ brings the resource in the idle state I_ℓ; through the working mode events wmk_j^i (raised by the partial processes DLUs) the resource moves to the busy state B_ℓ, which can be split in more states if it is necessary to distinguish between different allocations; the fault event f_ℓ brings the resource in state F_ℓ, depending from the fault description it can occur when the resource is idle or when it is busy; finally event sb_ℓ turns the resource off to state SB_ℓ.

Once the resources automata are built the behaviour of each partial process has to be projected on the resources where it can be allocated. This is done simply taking into account the possibility of putting the process in standby on one resource and reallocating it to another one. The automaton representing the allocation of the i-th partial process on the ℓ-th resource is called K^i/ℓ. Finally all the automata representing the resources and the partial processes on the resources are composed using parallel composition to build the automaton G modelling the uncontrolled behavior of the system. On this automaton the specifications H are designed to build the controlled behaviour K of the whole system, given by the parallel composition of H and G.

The following steps summarize the algorithm to design the GRRM decision logic unit.

Step 1: Modelling images of partial processes on the resources. Aim of this step is to model replica of the automaton K^i associated to a partial process according to the allocation specifications. If a partial process i can be allocated on a resource ℓ, it is necessary to model the image of i on the ℓ. The new DES is called K^i/ℓ. To move the partial process from one resource to another in a reallocation policy due to a fault, the process has to be put in standby in one resource and reallocated on the other one. For this reason the automaton K^i/ℓ is enriched with a new state SB_ℓ^i and two controllable events called on_ℓ and sb_ℓ. If state SB_ℓ^i is active in K^i/ℓ then i-th partial process is not running on the ℓ-th physical resource. Events sb_ℓ and on_ℓ are used by the GRRM to move process from a resource to another. When event on_ℓ is raised by the GRRM, automaton K^i/ℓ moves form state SB_ℓ^i to initial state in K^i. If the event sb_ℓ is issued by the GRRM the automaton K^i/ℓ is moved back to standby state SB_ℓ^i.

Step 2: Building the global uncontrolled behaviour model. The automaton G modeling the global uncontrolled behaviour is built as the parallel composition of all the automata K^i/ℓ representing the images of partial processes and all the automata R_ℓ representing the physical resources.

Step 3: Designing the decision logic unit of the GRRM. The automaton representing the decision logic unit of the GRRM is designed following the above introduced requirements on process allocation, working modes overlapping and hierarchy, external signals. The obtained automaton H represents the specifications for G. This step is obviously strongly application-dependent.

Step 4: Building the global controlled behaviour model. Finally the controlled behaviour of the whole system can be computed as the parallel composition of G and H. The obtained automaton is denoted with K.

The EG unit of the GRRM receives all the activating signals from the EG units of the partial processes and the diagnostic results of the resource monitors. Based on this information and on the specifications in the DLU of the GRRM, it validates one of the activating signals or starts the reconfiguration of the resources. The new WM signal produced by the DLU of the GRRM is sent down to the lower level of the architecture.

It is worth noting once again that the reconfiguration policy is not really distributed in the sense of the decision taking. Anyway the design of the reconfiguration specifications and the detection of reconfigurable functions are performed in a distributed way. The reason for this choice is that when multiple faults are considered and multiple processes are running in parallel, reconfiguration decisions cannot be taken without knowledge of the whole process, unless (and this is the case of the above introduced procedure) a first rough decision is taken and then validated at a higher level.

3.4 An application: the two-tanks system

In this section an application of the procedure introduced in Sec. 3.3 is presented. The plant used to this purpose is a two-tank system, modelled on the real plant located at the Centre de Recherche en Automatique de Nancy (CRAN), France. The results here presented are partially based on the work in [15].

The system (see [35] for a complete description) is composed by two tanks supplied by two pumps (P1 and P2) with flow rates Q_1 and Q_2. The two tanks are connected through two redundant pipes with valves V_{12} and V'_{12}. The two pipes are placed over the lower limit level of liquid and at the same height. The output flows of the two tanks are mixed through valves V_{F1} and V_{F2}. A schematic representation of the system is shown in Fig. 3.11.

The system is equipped with two level sensors (S_{L1} and S_{L2}) measuring liquid heights (L_1 and L_2) in the tanks and five sensors measuring flows Q_1, Q_2, Q_{12}, Q_{F1} and Q_{F2}. Considering valve V'_{12} closed in nominal situation, the physical equations of the system are

$$\begin{aligned} S_1\dot{L}_1 &= -Q_{F1} - Q_{12} + Q_1 \\ S_2\dot{L}_2 &= -Q_{F2} + Q_{12} + Q_2 \end{aligned} \tag{3.10}$$

where

$$Q_{12} = sign(L_1 - L_2)R_{12}\sqrt{|L_1 - L_2|}\,, \tag{3.11}$$

$$Q_{F1} = R_1\sqrt{L_1} \tag{3.12}$$

$$Q_{F2} = R_2\sqrt{L_2} \tag{3.13}$$

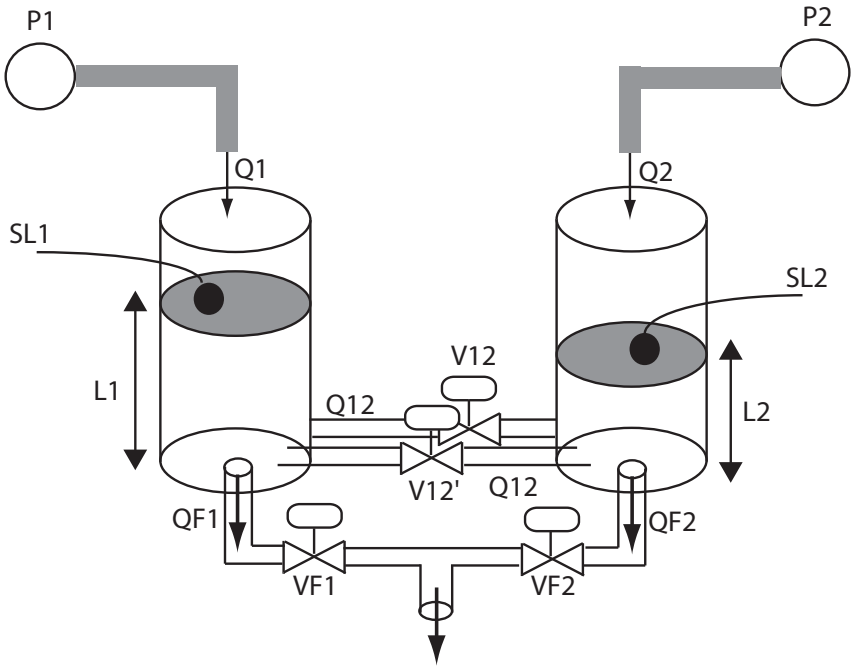


Figure 3.11: Two-tanks representation.

where R_{12} is the throttling of valve V_{12}[1], R_1 is the throttling of valve V_{F1}, R_2 is the throttling of valve V_{F2} and S_1, S_2 are the sections of tanks 1 and 2 respectively. From equations (3.10) to (3.13) the complete mathematical model of the system is then

$$\begin{aligned} S_1\dot{L}_1 &= -R_1\sqrt{L_1} - sign(L_1 - L_2)R_{12}\sqrt{|L_1 - L_2|} + Q_1 \\ S_2\dot{L}_2 &= -R_2\sqrt{L_2} + sign(L_1 - L_2)R_{12}\sqrt{|L_1 - L_2|} + Q_2. \end{aligned} \tag{3.14}$$

3.4.1 Control objectives and strategies

Flows Q_1 and Q_2 are considered as control inputs of the system. The controlled outputs are the total output flow y_1 and the mixing y_2 of the flows.

$$\begin{aligned} y_1 &= Q_{F1} + Q_{F2} = R_1\sqrt{L_1} + R_2\sqrt{L_2} \\ y_2 &= \frac{Q_{F1}}{Q_{F2}} = \frac{R_1\sqrt{L_1}}{R_2\sqrt{L_2}}. \end{aligned} \tag{3.15}$$

These two outputs are required to follow two desired set points, denoted respectively as y_1^* and y_2^*. They can be rewritten as desired set points L_1^* and L_2^* for the measured levels L_1 and L_2. In fact from definitions (3.15) we obtain

$$\begin{aligned} L_1^* &= \left(\frac{y_1^* y_2^*}{R_1(1 + y_2^*)}\right)^2 \\ L_2^* &= \left(\frac{y_1^*}{R_2(1 + y_2^*)}\right)^2. \end{aligned} \tag{3.16}$$

[1] V_{12} is an electromechanical valve and its throttling R_{12} is modelled as $k_1V + k_2$, where k_1, k_2 are suitably sized parameters and V is the applied electrical voltage.

Two nominal working conditions for the system are considered, assuming that the throttling of valve V_{12} is constant: the first one in which the valve is closed (decoupled system), the other one in which the valve is opened (coupled system).

In the first situation (decoupled tanks) model (3.14) becomes:

$$\begin{aligned} S_1\dot{L}_1 &= -R_1\sqrt{L_1} + Q_1 \\ S_2\dot{L}_2 &= -R_2\sqrt{L_2} + Q_2 \ . \end{aligned} \tag{3.17}$$

In case of valve opened, the model of the system is represented by (3.14). The system is therefore coupled and must be controlled in a coupled way in order to achieve again

$$L_1 = L_1^* \qquad L_2 = L_2^*$$

where L_1^* and L_2^* are defined in (3.16).

3.4.2 Fault scenario and first decomposition

To cover some significant detection and reconfiguration possibilities for this system five realistic faults are considered. All faults are supposed to occur in tank 2, anyway analogous faults can be considered for tank 1.

- *Actuator fault*: pump 2 can stuck to a constant value $Q_2 = \overline{Q}_2$, namely it injects a constant flow in tank 2. This fault leads to a loss of controllability as it changes one of the structural constraints in (3.14).
- *Loss of power of pump 2*: pump 2 can be also affected by a loss of power, modelled by a multiplicative (unknown) term δQ_2, which modifies the dynamics (3.10) as

$$\begin{aligned} S_1\dot{L}_1 &= -Q_{F1} - Q_{12} + Q_1 \\ S_2\dot{L}_2 &= -Q_{F2} + Q_{12} + \delta Q_2 \cdot Q_2 \, . \end{aligned}$$

- *Hardware fault*: valve V_{12} can stuck to a constant value $R_{12} = \overline{R}_{12}$ which can be 0 (valve closed) or a constant finite positive value. Note that a constant input throttling value is always given to the controlled system (both in coupled or decoupled working modes, because in the second situation we give to the throttling a 0 value), so if valve is stuck to that input value the system always behaves in the same way. Then the only relevant faulty case occurs when the system is working in a coupled mode and the valve is stuck closed, i.e. an electrical fault occurs and the voltage given to the valve becomes 0.
- *Leakage fault*: namely a hole in tank 2, i.e. there is an undesired outgoing flow from tank 2; the dynamic of L_2 is then corrupted by a term

$$\delta Q_{F2}(L_2) = h \cdot L_2$$

  where h is the section of the hole.
- *Level sensor fault*: the measure L_{2m} of level L_2 can be corrupted by a constant bias δL_2, i.e.

$$L_{2m} = L_2 + \delta L_2.$$

Due to the considered faults a first functional analysis of the system leads to a first decomposition in a Fault Tolerant Control (FTC) module achieving fault tolerant control of tank 2 and a Fault Tolerant Measurement (FTM) module managing fault tolerant measure of the liquid level in tank 2 as represented in Fig. 3.12. The interconnection valves are considered as redundant hardware supervised by a Resource Monitor achieving FDI.

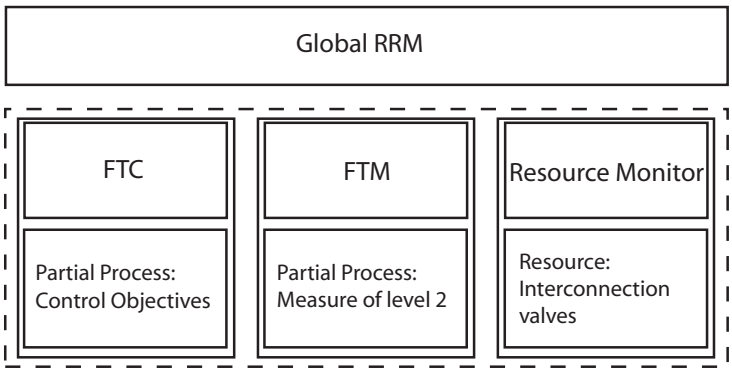


Figure 3.12: Two tanks basic decomposition.

3.4.3 Modular Diagnosis

A part of the structural graph of the system is represented in fig. 3.13 where the part related to L_1 has not been represented because it is a replica of the part related to L_2. In the structural graph some measurements and all control variables (and laws) have also not been represented.

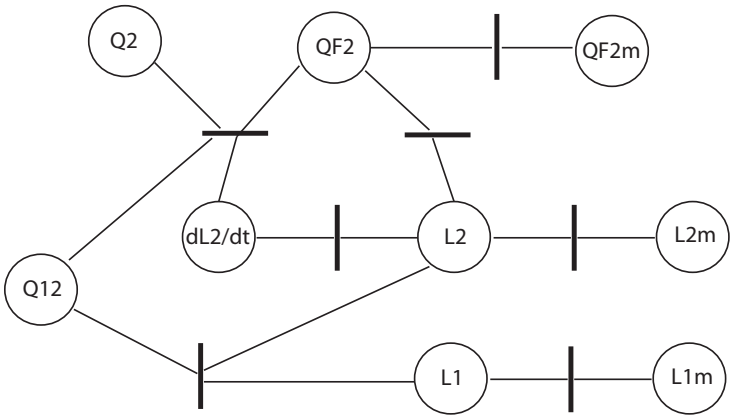
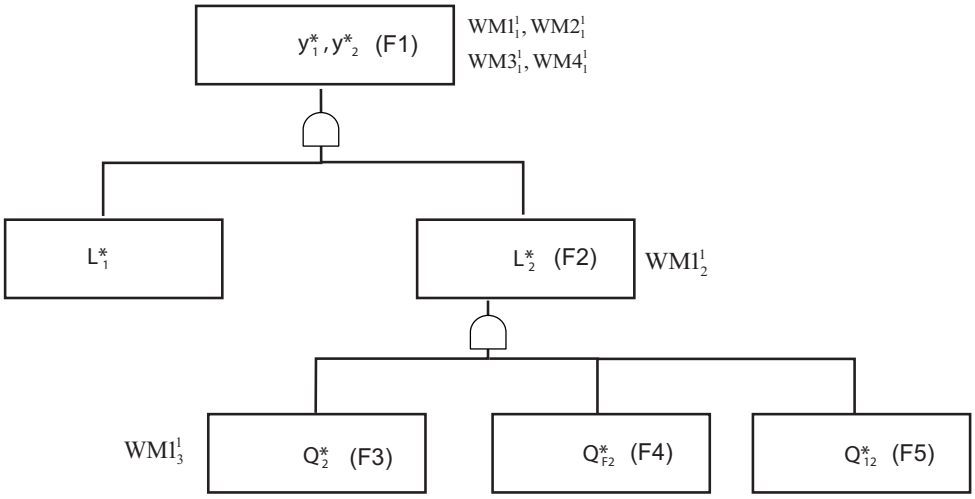


Figure 3.13: Two tanks partial structural graph.

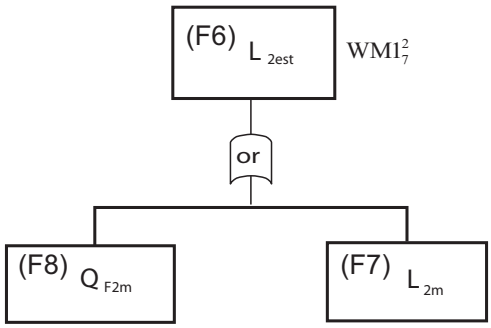
After applying the procedure of Sec. 3.1.3, the functional tree of the FTC module is obtained (fig. 3.14(a)). In this figure the main functionality (F1) represents the capability of tracking both objectives $y_1^\star$ and $y_2^\star$ (see def. (3.15). This is obtained if both levels L_1 and L_2 track the respective set-points (3.16). Sub-nodes are generated only from level 2 since only faults in tank 2 are considered. Level 2 is maintained at the desired value (functionality F2) only if the outgoing flow from tank 2 has the expected value $Q^\star_{F2}$ (functionality F4), the input flow of tank 2 has the desired (control) value $Q_2^\star$ (functionality F3) and the interconnection flow has the expected value $Q_{12}^\star$ (functionality F5).

The functional tree of the FTM module is represented in fig. 3.14(b). From relation (3.13) it is possible to see that the main functionality (estimation of the level in tank 2, F6) can be obtained through the measure L_{2m} of the level itself (functionality F7) or through its computation using the measure of Q_{F2} (functionality F8).

For the application of the presented procedure some residual signals able to detect loss of



(a) Functional tree for the FTC module.



(b) Functional tree for the FTM module.

Figure 3.14: Functional analysis for the two tanks system.

functionalities have to be introduced. Starting from the equations of the system (3.14) some residual signals partially based on structural considerations are generated. Nonlinear observers for L_2 are designed using the second equation of system (3.10) and the relations (3.11), (3.13) linking flow rates to the liquid level.

A first possible observer is based on the reliable measures of system variables and it is given by

$$S_2\dot{\hat{L}}_2 = -R_2\hat{L}_2 + Q_{12m} + Q_{2m} + G(\hat{L}_2 - L_{2m}) \ ,$$

where G is a suitable negative gain. The residual signal is then

$$r_1 = |\hat{L}_2 - L_{2m}| \ . \tag{3.18}$$

Residual r_1 is able to detect differences between the measure and the estimation of variable L_2. Hence the loss of functionality F7 (*measure of* L_2) is observable through r_1.

Another possible observer can be obtained by using measures and the controlled input flow value $Q_2^\star$, as

$$S_2\dot{\hat{L}}'_2 = -Q_{F2m} + Q_{12m} + Q_2^\star + G\,'(\hat{L}'_2 - L_{2m}) \ ,$$

where $G\,'$ is a suitable negative gain. From this a further residual signal can be designed as

$$r_2 = |\hat{L}'_2 - L_{2m}| \, . \tag{3.19}$$

Residual r_2 relies on the measure of outgoing flow Q_{F2} and on the comparison between the desired incoming flow in tank 2 ($Q_2^\star$) and the real actuated quantity Q_2. For this reason (see fig. 3.14(a)) this residual is able to observe both the loss of the *expected outgoing flow* functionality (F4) and the loss of the *desired incoming flow* functionality (F3). Then it can be associated to the loss of functionality F2 (*tracking of* L_2), because functionality F5 is considered to be always reliable due to the hardware redundancy of the interconnecting valve (managed by the GRRM at the higher level). Moreover, considering the way in which r_2 is generated, it is able to observe loss of functionality F7 (*measure of* L_2) and loss of functionality F8 (*measure of expected outgoing flow* Q_{F2}). Hence, from fig. 3.14(b), residual r_2 can also be associated to loss of functionality F6 (*fault tolerant estimation of level* L_2).

Due to the structural properties of the system, constraints linking throttling of valve V_{12} to other variables depend on the state of the valve itself. If the valve is open, it turns out that the presence of the flow Q_{12} between the two tanks and the measure of L_1 and Q_{12}, allow one to estimate L_2 as (see eq. (3.11))

$$\hat{L}''_2 = L_{1m} - \frac{|Q_{12m}|}{R_{12m}}$$

and to generate a residual signal r_3 as

$$r_3 = |L_{2m} - \hat{L}''_2| \tag{3.20}$$

which is linked to the loss of functionality F7 (*measure of* L_2).

It is necessary to consider the different algorithms used to estimate the sensor faults and to reconstruct the level measure as characterized by different reliability factors. Indeed it is expected that the algorithm to be run in the case of valve opened has an higher confidence level with respect to the ones to be considered if the valve is closed as it is based on algebraic physical relations.

	$\overline{Q}_2$	δQ_2	δQ_{F2}
r_2	1	1	1
r_4	0	0	1
r_5	0	1	0

Table 3.4: Residual matrix for the fault tree in fig. 3.15(a)

	$\overline{Q}_2$	δQ_2	$\overline{R}_{12}$	δQ_{F2}	δL_2
r_1	0	0	0	0	1
r_2	1	1	0	1	1
r_3	0	0	1	0	1
r_4	0	0	0	1	1
r_5	0	1	0	0	0

Table 3.5: Residual matrix for the two-tanks system

Consider now equation (3.13) which reveals that, in nominal conditions, the outgoing flow from tank 2 depends only on level L_2 and on throttling R_2 of the outgoing valve. In case of leakage in tank 2 this relation modifies as

$$R_2 L_2 = Q_{F2} + \delta Q_{F2}$$

which suggests to design a fourth residual signal as

$$r_4 = R_2 L_{2m} - Q_{F2m} \tag{3.21}$$

able to detect the loss of functionality F4 (*expected outgoing flow*) or loss of functionality F6 (*estimation of level* L_2).

Finally a probing test can be performed over the pump 2 in order to generate a residual signal r_5 able to observe a loss of functionality F3.

Starting from the functional trees and by bearing in mind the previous discussion, it is possible to build the fault trees. For the FTC module the fault tree is represented in fig. 3.15(a) and for the FTM module in fig. 3.15(b).

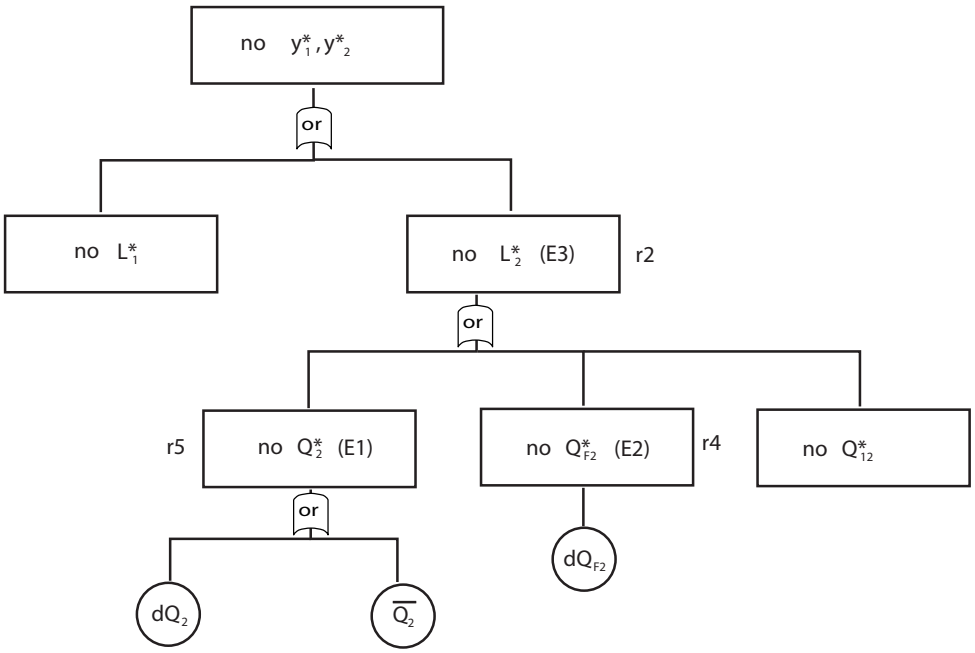
These fault trees are useful to design a residual matrix following a modular construction as described in Sec. 3.3. The residual matrix associated to the main loss of functionality of the fault tree in fig. 3.15(a) is reported in table 3.4: the loss of functionality $E3$ is detected if r_2 or r_4 or r_5 rises to 1, but it is also useful to distinguish between loss of functionality $E2$, detected only if r_4 rises to 1, and loss of functionality $E1$. Then loss of functionality $E1$ is detected only if r_5 rises to 1. This nested interpretation of the residual matrix allows hierarchic inference algorithm to detect failures. The same interpretation can be performed to obtain, from the FTM fault tree of fig. 3.15(b), the residual matrix associated to the FTM fault tree (not reported here for the sake of brevity).

The global residual matrix is presented in Table 3.5 and it can be obtained by composing residual matrices for FTC module fault tree and for FTM module.

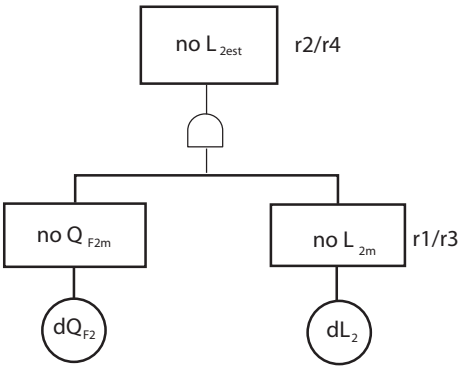
Note that diagnosis of fault $\overline{R}_{12}$ is not considered in this analysis because managed by the resource monitor.

3.4.4 Modular Reconfiguration

Next step is to detect reconfigurable functionalities and to associate to each of them some reconfiguration actions, namely some working modes.



(a) Fault tree for the FTC module.



(b) Fault tree for the FTM module.

Figure 3.15: Fault tree analysis for the two tanks system.

The top functionality F1 in fig.3.14(a) is reconfigurable. Indeed if controllability properties are lost and the tracking objectives cannot be simultaneously achieved, it is possible to compute degraded references trajectories, consistent with the faulty conditions, and to force relaxed tracking objectives to preserve the main tracking functionality but not the desired original objective.

Going to lower levels of the actual functional tree functionality F2 is also reconfigurable in presence of a leakage inducing a loss of the functionality F4: the presence of a leakage introduces an additive disturbance δQ_{F2} which can be compensated by suitably switching to a robust controller for the input Q_2 thus preserving the desired reference $L_2^\star$.

Similarly functionality F3 linked to the capability of regulating Q_2 can be considered reconfigurable in presence of a loss of power in the second pump using a robust controller.

Consider now the functional tree in figure 3.14(b). In this case the top functionality F6 turns out to be reconfigurable because it is always possible to provide a reliable estimation of L_2 either using direct measurement or structural observation in presence of the sensor fault.

It is now possible to identify the working modes related to each reconfigurable function. In nominal conditions the $j-$th functionality in the i-th partial process is characterized by a nominal working mode which is denoted by $\mathbf{WM0}_j^i$ in the case the two tanks are decoupled and $\mathbf{WM0'}_j{}^i$ otherwise. Then the reconfiguration policy associated to each reconfigurable functionality can be specified as follows.

(F1) This reconfiguration is needed whenever pump 2 is stuck to a constant value, i.e. $Q_2 = \overline{Q}_2$ =const, so that only one of the two tracking objectives can be enforced. The possible remedial actions depend on the specific objective which must be preserved ($y_1 = y_1^\star$ or $y_2 = y_2^\star$) and on the state of the connecting valve V_{12}. If it is closed, the level in tank 2 stabilizes to a constant level $\overline{L}_2$ which is measured. Hence, if it is required to preserve $y_1 = y_1^\star$, a new reference trajectory $L_1^\star$ for level 1 is computed as

$$L_1^\star = y_1^\star - \overline{L}_2 \, .$$

Otherwise, in case priority is given to the second objective $y_2 = y_2^\star$, a new reference trajectory $L_1^\star$ is computed as

$$L_1^\star = y_2^\star \overline{L}_2 \, .$$

We label the two possible working modes associated to the two above scenarios as $\mathbf{WM1}_1^1$ and $\mathbf{WM2}_1^1$ respectively. In both cases the control structure does not change but only the reference signal $L_1^\star$ is updated.

If valve V_{12} is open, the system behaves as a two-dimensional system with one input Q_1 forced by a constant disturbance $\overline{Q}_2$. In this case a suitable controller can be designed able to get rid of the disturbance $\overline{Q}_2$ and forcing the controlled output $L_1 + L_2$ to the reference $y_1^\star$ if the first objective ($y_1 = y_1^\star$) must be preserved, or the controlled output L_1/L_2 to the reference $y_2^\star$ otherwise. We denote the two possible working modes as $\mathbf{WM3}_1^1$ and $\mathbf{WM4}_1^1$ respectively.

The automaton G_1^1 representing functionality F1 is then reported in fig. 3.16, where it is possible to notice that after the failure occurrence all the reconfiguration possibilities are open. The reconfiguration specifications H_1^1 associated to the DLU of F1 are designed in fig. 3.17 where the activating events now separate the reconfiguring working modes.

(F2) This functionality can be reconfigured by switching to a robust controller on the input Q_2 able to get rid of the constant additive disturbance δQ_{F2} in presence of leakage. The corresponding working mode is denoted by $\mathbf{WM1}_2^1$.

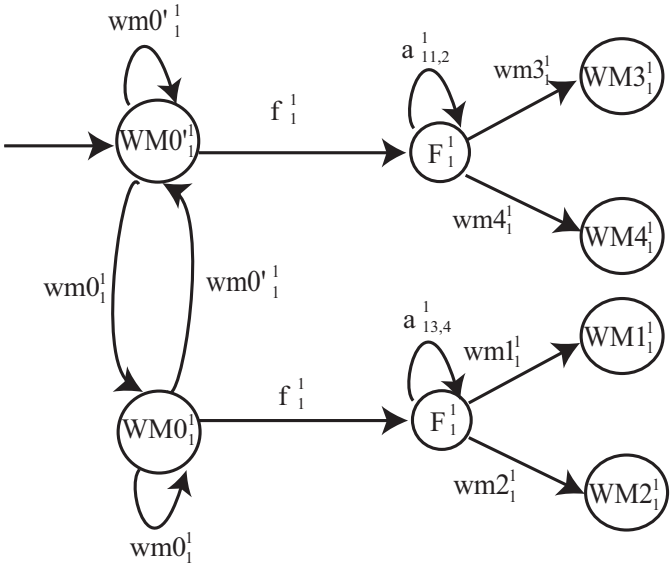


Figure 3.16: Automaton G_1^1 representing functionality F1.

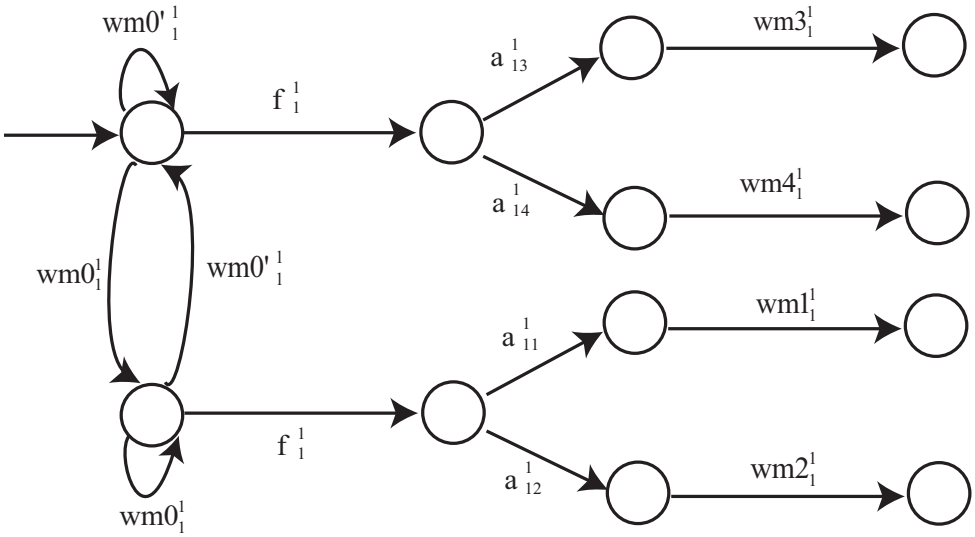


Figure 3.17: Reconfiguration specifications H_1^1 for functionality F1.

The reconfiguration specifications H_2^1 for F2 are represented in fig. 3.18. For the sake of brevity the obvious automaton G_2^1 is not reported here.

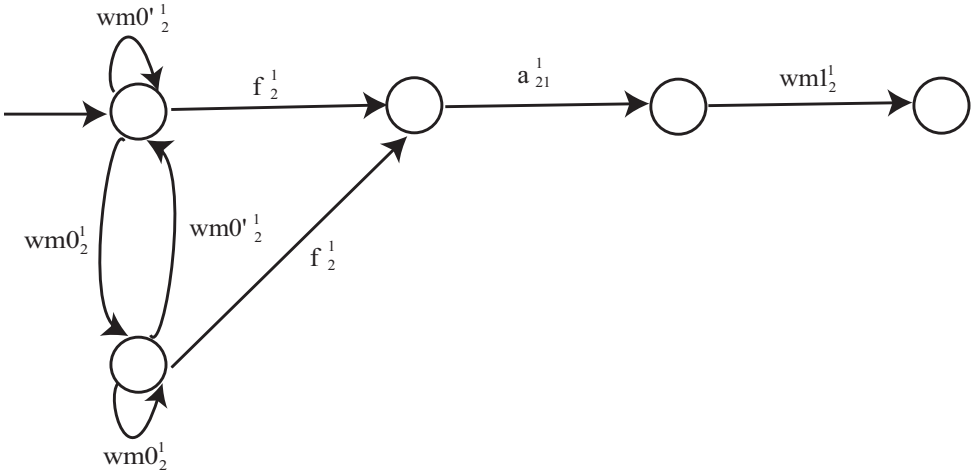


Figure 3.18: Reconfiguration specifications H_2^1 for functionality F2.

(F3) This functionality is reconfigured through the nominal controller on the pump 2 which is supposed to be robust with respect to uncertainties affecting the input gain (induced by the loss of power fault). The working mode associated to this implicit reconfiguration is denoted by $\mathbf{WM1}_3^1$.

The reconfiguration specification H_3^1 are depicted in fig. 3.19. Again for the sake of brevity automaton G_3^1 is not reported.

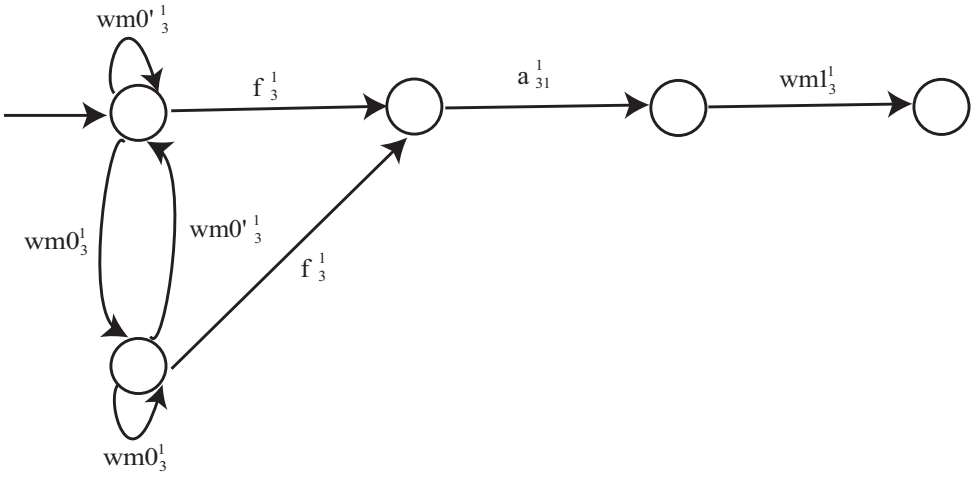


Figure 3.19: Reconfiguration specifications H_3^1 for functionality F3.

(F6) This functionality can be simply reconfigured by substituting the level measure with a suitable estimate, obtained through one the methods presented before, once a sensor fault is detected. We label the associated working mode as $\mathbf{WM1}_6^2$.

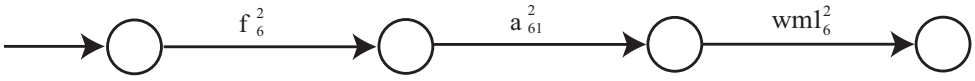


Figure 3.20: Reconfiguration specifications H_6^2 for functionality F6.

	$\overline{Q}_2$	δQ_{F2}	δQ_2 (small)	δQ_2 (severe)
a_1^{11}	1	0	0	1
a_1^{12}	1	0	0	1
a_1^{13}	1	0	0	1
a_1^{14}	1	0	0	1
a_1^{21}	0	1	0	0
a_1^{31}	0	0	1	0

Table 3.6: Look-up table describing the Event Generator of partial process 1

Fig. 3.20 shows the reconfiguration specifications H_6^2 for function F6. Note that for partial process 2 there is no need to have two nominal working modes because they are related to the different ways the system is controlled (depending on the coupling of the two tanks), but not on the way the estimate of level in tank 2 is obtained.

The event generator of partial process 1 is here represented by a look-up table (see Table 3.6) mapping the occurred faults to the activating events previously introduced.

It is important to note that fault δQ_2 involves the activation of two kinds of events: a_{31}^1 if the fault is of small entity, $a_{11,2,3,4}^1$ if the fault is severe. Indeed δQ_2 can be implicitly accommodated (reconfigurable functionality F5) or can be treated as a stuck fault (reconfigurable functionality F1). The choice is made by the EG elaborating the estimation of the fault according to a performance criterion.

3.4.5 Design of the GRRM

The whole system is supervised by a Global RRM used to manage the hardware reconfiguration, i.e. the fault on the interconnection valve: if the valve is stuck closed we can switch to valve V_{12}' and use R_{12}', due to the parallel hardware redundancy. This working mode is denoted with **WMR**.

First the redundant valves are modeled as automata (fig. 3.21). Valve V_{12} is in idle state (I) until a working mode $\mathbf{WMk}_j^i$ is issued. In this case the valve is busy (B) in the sense that it is the valve the system is using. When fault $\overline{R}_{12}$ occurs the valve moves in faulty state (F). Event sb is used to force the valve in stand-by and move the automaton from any state to stand-by state (SB). From this state event on can be used to enter again idle state.

The DES modelling valve V_{12}' is normally in stand-by state (SB') unless event on' in enabled. In this case the new state is the idle state (I') and the working mode **WMR** is imposed. Valve V_{12}' is now in busy state (B') (the valve is in use) and any working mode $\mathbf{WMk}_j^i$ can be issued. Event sb' is used to bring back the valve in stand-by state.

Then automata K^i have to be allocated on the resources. Due to the properties of parallel composition, refining K^i and refining each H_j^i and G_j^i leads, after the composition of all H_j^i, G_j^i over the ℓ-th resource, to the same result. This let us show just automaton H_j^i/ℓ avoiding the representation of K^i/ℓ which may explode.

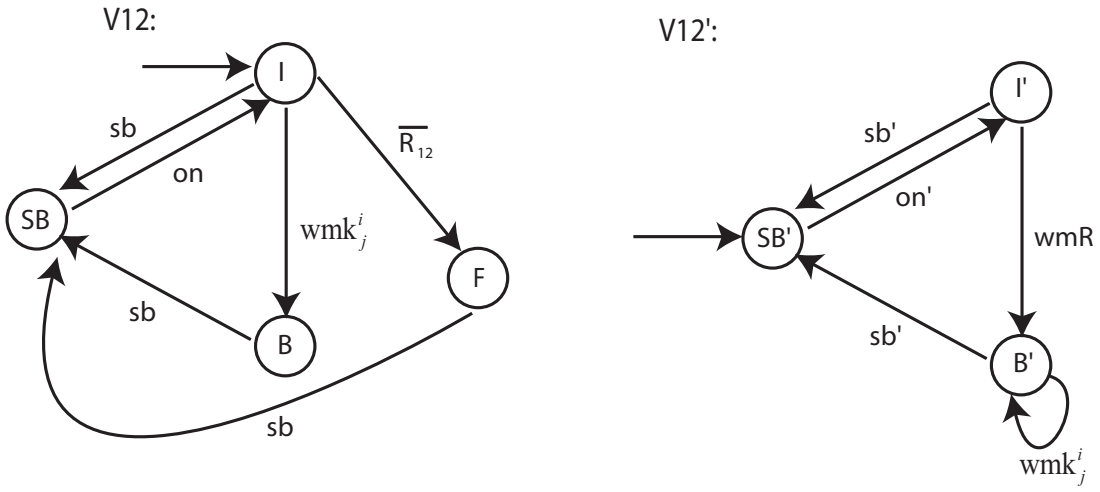


Figure 3.21: Valves V_{12} and V'_{12} as DES.

In fig. 3.22 and fig. 3.23 the specification H_3^1 allocated over valve V_{12} and V'_{12} respectively is depicted. Note the use of events *sb*, *on*, *sb′* and *on′* to move the working modes allocation from one resource to the other. It is easy to extend this procedure to the other specifications.

Finally the behavior of the Global RRM is obtained and shown in fig. 3.24. When the interconnection valve V_{12} is stuck closed, the supervisor forces this valve in stand-by and switches to the other redundant valve V'_{12} imposing working mode **WMR**.

3.4.6 Implementation and Experimental Results

To demostrate the effectiveness of the proposed solution, some significative experiments have been performed on the real plant. The supervisor decision logic and the controllers have been realized in Simulink and than implemented in *ControlBuild* in collaboration with *TNI*. To avoid bumps due to the switch among controllers, they have been factorized in order to have a state-space representation with shared state.

The system is linearized in point: $(L_{10}, L_{20}) = (0.4\,,\, 0.5)\ m$. As a test scenario the outputs of the system must follow a set-point which changes both in nominal and faulty conditions. The system starts in nominal conditions with constant input values: $y_{10} = 1.4669 \cdot 10^{-4}$, $y_{20} = 0.8338$.

At time $t = 1000\ sec$ a step set-point is given to the system for both objectives with final values: $y_1^* = 1.4656 \cdot 10^{-4}$, $y_2^* = 0.9532$. At time $t = 2000\ sec$ a fault $\overline{Q}_2$ on pump 2 occurs and the controller is reconfigured forcing objective y_1^*. At time $t = 3000\ sec$ the set-points come back to nominal values. The system is still in a faulty working mode and follows the set-point by finding another steady-state point. The fault is accommodated at time $t = 4000\ sec$.

In fig. 3.26(a) the tracking of the first objective set-point is represented by the noisy signal, the values of set-point are represented by the straight line. When the set-point changes, both in nominal and in faulty conditions, it is possible to see that the controllers continue to track objective y_1.

In fig. 3.26(b) the tracking of the second objective set-point is again represented by the noisy signal, the value of set-point by the straight line. When the set-point of objective y_2 changes, the nominal controller tracks it because the fault is not occurred yet; when the fault occurs the

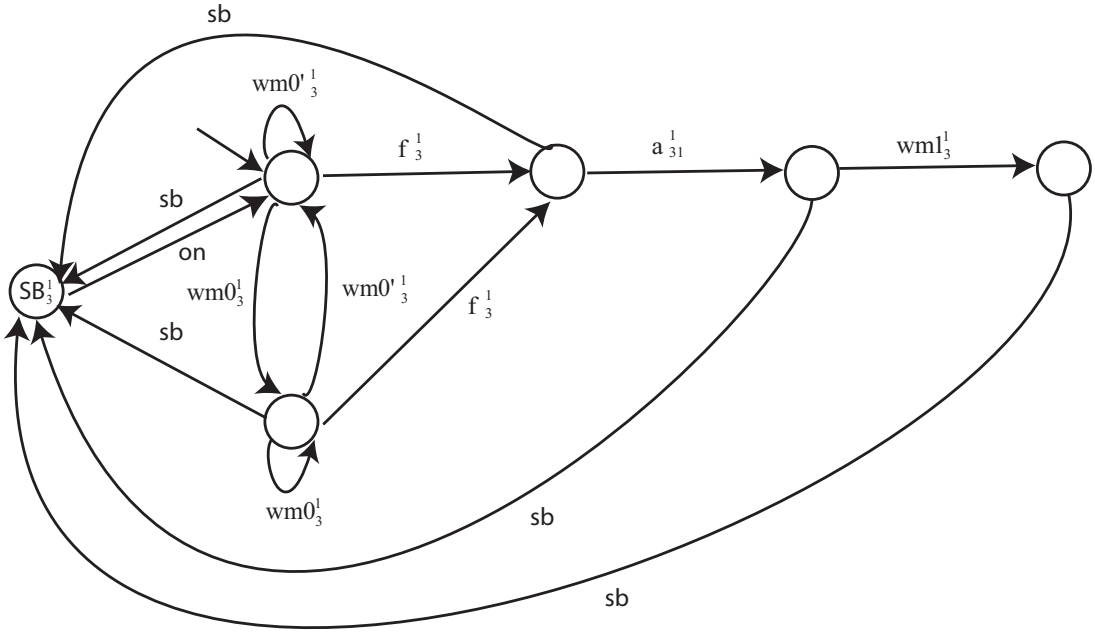


Figure 3.22: Specifications H_3^1 over V_{12}.

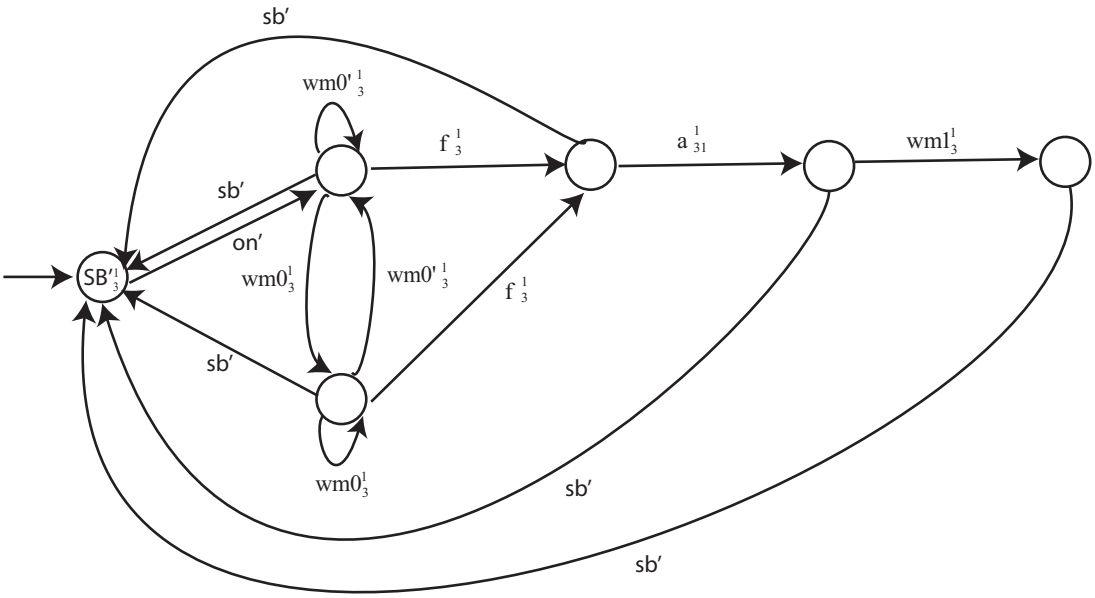


Figure 3.23: Specifications H_3^1 over V_{12}'.

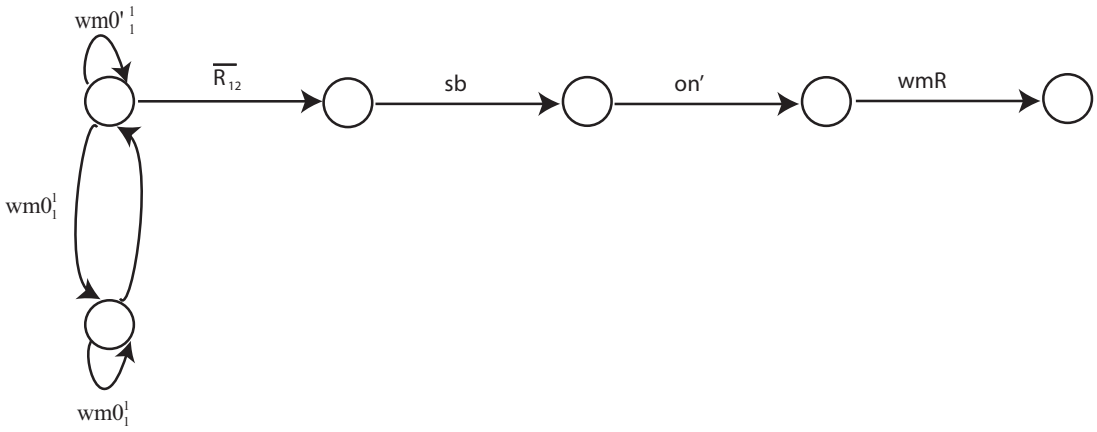


Figure 3.24: Reconfiguration specifications for the GRRM.

Figure 3.25: The real plant.

second objective is not achieved anymore because the forced objective is y_1. After the fault accommodation, objective y_2 is tracked again due to the switch back to nominal controller.

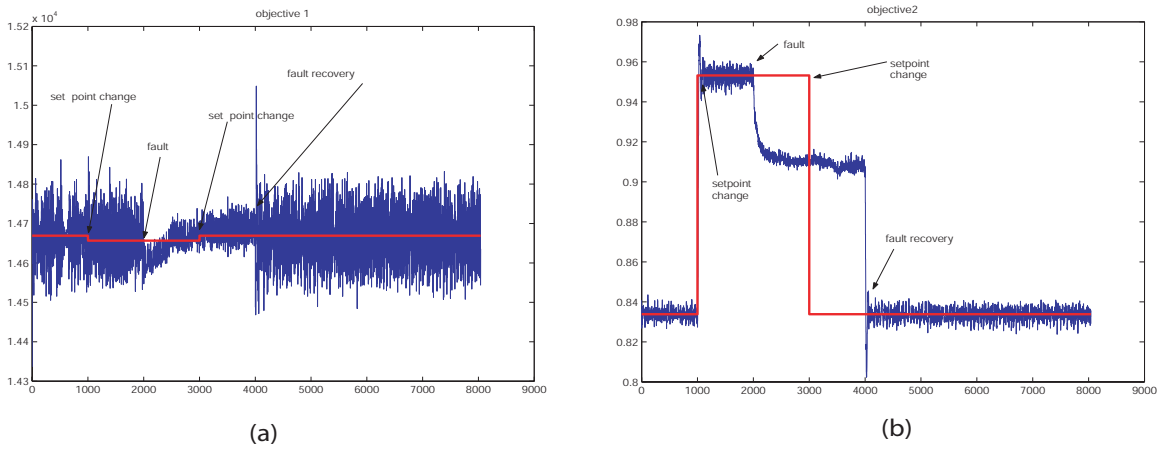


Figure 3.26: Objective y_1 (a) and y_2 (b) with fault on pump 2 at time $t = 2000$ *sec* and set-point changes. The controller is reconfigured forcing objective $y_1^\star$ ($\mathbf{WM3}_1^1$) Set-point values are represented by the straight line.

Chapter 4

Fault Detection for Hybrid Systems

In Chapter 3 distributed systems have been analysed from an high level point of view. This has lead to the use of discrete event systems for their reconfiguration, because only the discrete changes given by the faults were taken into account in the architecture of section 3.2.

This chapter presents a deeper picture of the nature of large scale system. Basically, distributed systems as they were considered in the previous chapter can be viewed as hybrid system (see chapter 2) where the discrete part is given by the different working modes and the continuous part is given by the system dynamics in each working mode.

A more general problem is the extension of the fault tolerance methods to hybrid systems. As presented in section 2.3 there are many methods for modelling hybrid systems, and the choice of one of them depends on the system property to be studied. For this reason when dealing with fault tolerance the capability of modelling the different kinds of faults that can occur in the system depends on its representation.

In the first part of this chapter a critical analysis of the faults which is possible to represent in the different modelling frameworks is presented.

It will be shown that not many attempts have been made until now in the field of fault tolerance for hybrid systems. This can be due in first instance to the hard task of state estimation in this kind of systems. Indeed to know if a fault has occurred it has to be detected if the system is behaving in an unusual way, that is based on the knowledge of the state in which the system is working. When dealing with hybrid systems a state estimator must provide both the continuous and the discrete state. The accomplishment of this task is generally difficult because of the coupling of the two dynamics. This means that a good estimator must provide the two states simultaneously. The estimation algorithms provided until now are mostly based on Moving Horizon Estimation and Particle Filtering for the continuous state and Hidden Markov Models for the discrete state. With the help of these estimators, the considered faults are usually treated as additive noise to the system.

When the parameters of the new faulty system are not known as well, system identification comes into the play. As introduced in chapter 1 and in [39] for continuous-variable dynamical systems, model based fault diagnosis starts from the model of the system. System identification methods for fault diagnosis are an alternative way to estimation methods, when the parameters of the faulty system have to be identified and/or when the fault is incipient, hence it can be seen a small change in the parameters. For this reason in this chapter some system identification methods for hybrid systems will be recalled and compared, and some new issues for their application to fault tolerance presented.

An interesting extension to all these results is represented by the introduction of distributed hybrid systems. This kind of systems has been introduced to better deal with the actual complex

systems, which are more and more constituted by heterogeneous systems with different functionalities but connected through a network to fulfill a global task. Some modelling frameworks and design/analysis methodologies have been reviewed to deal with this kind of systems too. The main changes are due to the fact that dealing with distributed systems the necessity to reduce the computational burden increases. For this reason the methodologies used with this kind of systems try to exploit their distributed nature as much as possible. Then some decentralized methods have been used to cope with state estimation and fault detection in distributed hybrid systems.

The most typical kind of faults which can occur in distributed systems are connection faults, i.e a kind of communication faults. These faults should carefully be detected and reconfigured because they can lead to propagation of the fault effect from one sub-system to another. A connection fault always means a loss of information. This loss can be due to a complete breakdown of the connection, which means that no data would be available from one sub-system to the other. More precisely a connection fails when the sending sub-system sends some data that the receiving sub-system does not receive. This means that a signal starts from the sender, but to the receiver no input signal arrives. This kind of faults can be modelled in a distributed hybrid system in two ways. We can only take into account the signals and use input and output signals of the sub-systems to connect them. A second option is to consider the connections ad hybrid systems, then they could be modelled as hybrid sub-systems as well. In this second case some composition rules should be stated. Some modelling issues on this topic will be discussed in the last Section of this work.

4.1 Modelling Faulty Hybrid Systems

Faults and system modelling are really coupled for basically two reasons: the model used to describe a system influences the modelling of the kinds of fault that can occur in the system itself; the system model is decided based on the considered kinds of fault.

This second point means that if a fault forces a system in a new behaviour (say a faulty operating mode) then the system becomes hybrid because of the fault effect. Indeed if the fault changes some dynamics of the system, then a new operating mode is started, and some structural properties of the system itself can be changed. The mode changing reminds to the modelling of hybrid systems. Nevertheless a system can be hybrid by its nature, meaning that it can already have different operating modes, and hence hybrid dynamics. In this case the fault will not change the nature of the system itself, but will only introduce new dynamics that have to be modelled. And this leads to the first point of the above statement.

Therefore the most interesting nondeterministic effects for fault detection are mode changes. They are the effect of the new operative behavior of the system. They can occur due to autonomous switchings, i.e. depending on the continuous behavior, or due to control commands. In other words mode changes are originated by control switchings, reconfiguration policies and fault occurrence. In fault tolerance the knowledge of the different modes in which the system can work is extremely useful to know if a fault has occurred and to identify it. In fact to accomplish the diagnostic task and detect the fault occurrence three things are needed:

- the complete knowledge of the modes in which the system can work;
- the kind of faults which can occur in the systems;
- an estimation or identification algorithm able to determine in which of these modes the system is.

More precisely they are the basis for a diagnostic algorithm which is also able to identify the occurred fault.

Not many attempts have been made to apply some fault detection techniques to hybrid systems. However from the few literature on this topic it is possible to state that obviously even the choice of the FDI method is related to the considered model for hybrid systems.

Among the first methods for fault detection of hybrid systems it is worth citing the ones presented in [52] and [63]. These two methods are quite different, because they are based on opposite models of hybrid systems. The first one deals with mixed logical dynamical (MLD) systems, and mainly with faults on the continuous dynamics, whereas the second one uses quantised systems, then it deals mainly with the discrete part.

In [53] and [52] an application of Moving Horizon Estimation to MLD systems is introduced. This technique is also used for fault detection assuming that the model of the faulty system is known. Basically the considered faults in this framework are modelled as binary disturbances acting on the system inputs. This method is quite interesting because MLD can model a variety of different systems. However the fault modelling power is quite limited, restricted to the disturbance-like faults, and the computational burden is quite high due to the fault detection algorithm.

In [63] and [5] semi-Markov processes are used for fault detection in quantised systems. With this modelling framework almost any kind of fault can be considered, but their dynamics are assumed to be independent from the system state and input, their probabilities of occurrence known and they are modelled as quantised systems as well. The major drawbacks of this approach are the design effort to build the quantised representations and the approximation introduced by the quantisation. An interesting extension to this approach is reported in [47] where other diagnostic techniques are used, such as embedded maps, timed and nondeterministic automata. Finally some reconfiguration issues are treated in [49] where sensor and actuator faults are considered.

In [51] the authors consider hybrid system composed by continuous systems with a supervisory controller which gives the mode changes, but no autonomous switchings are allowed. The systems are modelled using a 5-tuple of describing elements, in a discrete event fashion. The considered faults are faults on components, abrupt, partial or complete. They are represented as exogenous events added to the controller actions. The diagnostic method is borrowed from the qualitative approach of AI, extending search techniques for continuous systems and using models comparison. With the proposed method the authors are able to diagnose if the system is in a faulty mode and in which one with respect to the fault effect on the system. This approach is quite effective, but they can model only a restricted number of systems and the considered faults are not well specified. Indeed it is more the effect of the fault which is taken into account and not the real physical fault.

A quite different approach has been introduced in [45], where the authors model hybrid systems as hybrid dynamic Bayesian networks. Thus they focus on systems that have linear dynamics given some particular values to the discrete variables. Moreover the systems they consider are composed by weakly interacting subsystems. They also claim that with this model it is possible to consider several different kind of faults, but they actually study only the case of abrupt faults, measurement errors and parameter drift faults. While abrupt (or as the authors call them, burst) fault can be represented with discrete changes in the system, measurement errors are usually considered in robustness problems. The presented approach uses Bayesian fault detection for the presented kinds of faults. The method represent a different point of view in the literature, but it seems restricted to a rather small number of systems and fault effects.

It is quite difficult in fault detection framework to state a general methodology to deal with

all kinds of faults in all kinds of systems. For hybrid systems this is even more complicated by the lack of unifying modelling theory, which leads the application of fault tolerant techniques even more application dependent. Nevertheless it is worth to look for a method able to deal with the highest number of possible faulty situations (remember the fault classifications in chapter 1), always depending on the model chosen to represent the class of hybrid systems to study.

Many other faulty situations can be figured out, depending on the knowledge of the mode in which the system is working and on the knowledge of the effect of the fault, i.e. the mode in which the system is moving. Finally it is impossible to know all the faults that can occur in a certain system, then we could take into account an unknown fault, i.e. an unknown operating mode. Some of these problems can be partially considered if the system is modelled with a Probabilistic Hybrid Automaton ([38]) or Stochastic Automata [19]. Generally speaking to consider, at least partially, these features some good estimation or identification method should be used. This will be the topic of Sections 4.3 and 4.4, whereas in next section two framework for FDI of hybrid systems are compared.

4.2 Comparison of two Techniques for Fault Detection in Hybrid Systems

In this section two methods for FDI of hybrid systems are compared. Even if both approaches are based on hybrid automata modelling formalism, the first one, introduced in [27], is more computer science oriented, while the second one, introduced in [21], is more control oriented. It is then interesting to compare them in order to highlight the main advantages and drawbacks of each approach.

The first method presents some results based on Hybrid Input/Output Automata (introduced in [50]) and extends the theory of diagnosability for discrete events systems to the hybrid case. As usual in this kind of discrete event approach to hybrid systems, the two dynamics are kept separated, which means that the diagnoser has to first check if some fault has occurred in the current (discrete) mode, then to check the continuous dynamics inside the mode, finally a supervisor will decide which kind of fault has occurred and where. Nevertheless the diagnosability is tested on the hybrid dynamics, using the notion on hybrid traces.

In the second approach the dynamics are considered all at once, i.e. the discrete dynamics are cast into the continuous dynamics definition. This means that the discrete dynamics are considered as switching and the differences from one mode to the other are only due to the continuous evolution of the variables inside each mode. The authors explain how it is possible to detect some kinds of discrete failures using their approach, in particular they are able to determine if the successor mode is different from the expected one. In this sense they do not use events and the fault consequences have always to be related to some changes in the continuous dynamics.

4.2.1 Fault Detection using Hybrid I/O Automata

One of the main advantages in using Hybrid I/O Automata (HIOA) is that almost all the theory already developed for the discrete event systems on automata (see 2.2) can be extended to the hybrid automata. The major drawback is that while doing this extension the notation becomes huge and a big effort should be made in formalizing the problems.

In [27] the authors start from the decomposition of the system, as shown in fig. 4.1, and build a hybrid input/output automaton for each component of the system. Then all the automata are composed using the composition procedure presented in [50] and the whole system is built. In

this framework the faults are seen both as discrete transitions from the nominal to the faulty state and as deviations of the continuous trajectories from the predefined set-points. This means that the faults can occur both in the discrete and the continuous dynamics.

Recalling the definition of HIOA given in Sec. 2.3 and referring to fig. 4.1, the automata representing the various parts of the system are:

- $P = (U_P, X_P, Y_P, \Sigma_P^{in}, \Sigma_P^{int}, \Theta_P, D_P, W_P)$ representing the main structure;
- $A_i = (U_i, X_i, Y_i, \Sigma_i^{in}, \Sigma_i^{out}, \Theta_i, D_i, W_i)$ representing the i-th actuator;
- the sensors S_i and the controller C are modelled as automata representing simple input/output maps;
- $H = (U_H, X_H, Y_H, \Sigma_H^{in}, \Sigma_H^{int}, \Theta_H, D_H, W_H)$ representing the plant;
- $S = (X_S, \Sigma_S^{int}, \Theta_S, D_S, W_S)$ representing the system.

The main assumption on the kinds of faults that can affect the system is that they do not affect the system output: the system produces the same output variables wether it is faulty or not. Then the fault occurrence comes out to be an internal action, which means that $\Sigma_P^{int} \neq 0$, whereas in nominal conditions $\Sigma_P^{int} = 0$. Since a fault can have consequences both in discrete and continuous dynamics, the information about the fault occurrence is given in two stages:

Continuous Stage: The continuous behavior is represented by W_P then a fault occurrence from this set is detected using analytical redundancy. When a fault is supposed to possibly happen the set U_P of input variables of the main structure can be partitioned in two sets: U_{PN} containing all the known inputs (such as control signals) and U_{PF} containing all the unknown inputs, i.e. the faults. The same for $X_P = X_{PN} \bigcup X_{PF}$. In this way it is possible to set:

- $V_{PF} = [U_{PF}\ X_{PF}]$ is the fault state vector, $\mathbf{V}_{PF}$ is the fault state space;
- $V_{PN} = [U_{PN}\ X_{PN}]$ is the no-fault state vector, $\mathbf{V}_{PN}$ is the no-fault state space.

Moreover different faults will lead to different fault modes: $\mathbf{V}_{PFj} \subseteq \mathbf{V}_{PF}$ where all $\mathbf{V}_{PFj}$ are pairwise disjoint, i.e. only one fault can occur at a time. The total state space will be: $\mathbf{V}_P = \mathbf{V}_{PN} \bigcup \mathbf{V}_{PF}$.

Discrete Stage: The discrete evolution of the state is represented by D_P. As for the continuous behavior it is possible to write $D_P = D_{PN} \bigcup D_{PF}$ with:

$$D_{PN} = \bigcup\{(s, \alpha, s') | (s, s') \in V_{PN}, \alpha \in \Sigma_P^{in}\} \subset D_P$$
$$D_{PF} = \bigcup\{(s, \alpha, s'') | s \in V_{PN}, s'' \in V_{PF}, \alpha \in \Sigma_P^{in}\} \subset D_P$$

The transitions in D_{PF} bring the main structure to fault state space $\mathbf{V}_{PF}$. As for the continuous case, also the discrete transitions have different kinds, one for each fault mode, which leads to

$$D_{PF} = \bigcup_{j \in E} D_{PFj}$$

where E is the set of all fault modes.

In [28] the authors present an analysis of diagnosability for hybrid systems, based on the definition of faulty guards:

$$Guard_F = \{v \in V | (v, \alpha, v') \in D_F \text{ for some } v'\}.$$

Hence the guards are defined in terms of variables. This means that there is a set V_g of variables associated to the guards such that:

$$V_g = V_{g\ ind} \bigcup V_{g\ d}$$

where $V_{g\ ind}$ are the linear independent variables, and $V_{g\ d}$ are the linear dependent variables. To the first ones the directly measurable guards are associated (given that the system is observable), while to the second ones the not-directly measurable guards are associated.

Finally a hybrid system S is diagnosable if it exists a hybrid trace (with any sufficient long continuation of alternation trajectories actions after the occurrence of a fault) that contains the same faulty guard, or in formal terms:

$$\forall F_i \in E, \exists k_i \in \mathbb{N}, \forall g_{Fi} \in \alpha_i, \forall \alpha_i \in S | \forall t \in \text{htrace}(a_i), ||t|| \geq k_i, g_{Fi} \in \text{htrace}(a_i).$$

A necessary and sufficient condition for diagnosability is also given:

Theorem 4.1. A hybrid system without multiple failures of the same mode is diagnosable if and only if:

1. There is a measurable faulty guard
2. No state in htrace(a) is *indistinct*, i.e. it is not clear which fault mode has occurred.

In [29] the authors present a possible implementation of the diagnoser using hybrid structure hypothesis tests.

The task of the diagnoser is to generate a signal whenever a fault occurs. To do this it has to compare the evolution of the system S with the acceptable behavior. Then when it detects a fault it generates a signal indicating the malfunctioning component.

In [29] fault diagnosis based on hypothesis tests is introduced. The main steps in this kind of diagnosis approach for hybrid systems are:

- for each fault a hypothesis test is designed
- since the faults can affect either discrete or continuous dynamics, two kinds of hypothesis tests are designed:
    1. a bank of discrete tests which provide the necessary information about the fault occurrence at this level of dynamics
    2. a bank of continuous tests which perform the continuous fault diagnosis (at the lower level)
- the tests are statistical and binary, i.e. there are two mutually disjoint hypothesis:
    1. if the fault belongs to the set of fault modes B_i (to be tested), then we have the null hypothesis H_0
    2. any different hypothesis is called alternative hypothesis H_1
- finally it has to be checked if the fault f_a is in B_i or in B_i^C (complement of B_i).

For the generation of the banks of tests, discrete and continuous, the interested reader is referred to [29].

Finally the diagnoser is designed as an hybrid automaton with as inputs: $u(t), \Sigma, y(t)$ and as output the decision statement S of the banks of tests. The diagnoser consists of three parts: the discrete diagnoser, the continuous diagnoser and the decision logic.

Discrete Diagnoser The aim of the discrete diagnoser is double: it has to perform the diagnosis of the faults at the discrete level and to estimate the discrete state. When an action occurs the diagnoser computes all the possible next discrete transitions, to each of them is associated the corresponding guard. It is checked if that guard is in the set of measurements guards and for each transition the hypothesis test is performed and the diagnosis sub-statement S_{Di} is generated.

Continuous Diagnoser The dynamics of the continuous diagnoser depend on the state of the hybrid system. Its inputs are the process input/output signals and the discrete diagnoser output. Its outputs are the diagnosis sub-statements S_{Cj}.

Decision Logic The final statement is given by: $S = \bigcap_i S_{Di} \bigcap_j S_{Cj}$.

4.2.2 Fault Detection of Hybrid Systems using Structured Parity Residuals

In [21] and [20] the model of hybrid systems used is a hybrid automaton:

$$< Q, X, Y, F, H, \sigma_s, \sigma_c >$$

where:

- Q is set of discrete states,
- X is the continuous state-space,
- Y is the output space,
- F is a finite set of functions defining trajectories of the continuous state $x(t)$,
- H is a finite set of functions defining the output variables relations with the state variables,
- σ_s defines spontaneous switchings from one mode i to another mode j, due to the intersection of the state vector with a jump surface $S_{ij} = x_i \in X_i \; s.t. \; s_{ij}(x_i) = 0$ where s_{ij} represents the switching condition,
- σ_c defines forced switchings due to external events $e_{ij} \in E$.

In this approach the choice of the model depends on the objective of the FDI procedure, i.e. if the model is used to isolate the faults or to diagnose of the faults. With this purpose the authors state that it is possible to use two models of the system:

- The normal operation model, representing the system in nominal situation, without faults; in this kind of models the procedures used are *a priori* able to detect faults, so they provide information on which constraint of the model is unverified such that isolation (location) of the faulty component is possible.
- The faulty operation model, representing the system in faulty situation, used when the faults have to be diagnosed (identified).

In [20] some examples of different kinds of faults for hybrid systems are stated. These faults can in general affect the current mode behavior (continuous dynamics) or the trajectory of the discrete evolution (discrete dynamics).

Among the faults affecting the current mode it is possible to find:

- Faults corrupting the equality constraints (i.e. state and output equations); they can be view as additive terms to the differentials and algebraic equations representing state and output dynamics.
- Faults corrupting inequality constraints representing the state space; usually observers and identification techniques are used to estimate these faults.
- Faults unintentionally modifying the discrete state; they can lead to a mode change, then they are only apparent mode faults.

For the faults affecting the discrete evolution it is possible to find three different situations:

- The system moves to an unexpected mode (not allowed in nominal conditions); this is a controller fault.
- The system moves to an allowed mode but due to an external event, not due to the control policy.
- The system stays on a mode when it is expected to switch to a new one.

While all the FDI approaches require the knowledge of the mode in which the system is working, the FDI algorithm must also provide an estimate of the mode.

In [21] and [20] parity space approach is used for FDI of hybrid systems. It is based on analytical redundancy relations that can be obtained from the model equations eliminating the unknown variables. In the linear case this is achieved generating parity residuals. Considering linear dynamics in each mode i of the hybrid system modelled with its normal operation model, it is possible to generate structured residual using the classical parity space approach. The model of the continuous dynamics in mode i is then:

$$\begin{aligned}\dot{x}_i(t) &= A_i x_i(t) + B_i u_i(t) + G_i^1 d_i(t) + F_i^1 \phi_i(t)\\ y_i(t) &= C_i x_i(t) + D_i u_i(t) + G_i^2 d_i(t) + F_i^2 \phi_i(t)\end{aligned}$$

where ϕ_i is the faults vector and d_i is the disturbances vector to which the residual signals must be robust.

Then writing the successive time derivatives of the output y_i to a given order p_i:

$$\begin{aligned}\overline{y}_i^{p_i} &= OBS(C_i, A_i, p_i)x_i\\ &+ COM(A_i, B_i, C_i, D_i, p_i)\overline{u}_i^{p_i}\\ &+ COM(A_i, G_i^1, C_i, G_i^2, p_i)\overline{d}_i^{p_i}\\ &+ COM(A_i, F_i^1, C_i, F_i^2, p_i)\overline{\phi}_i^{p_i}\end{aligned}$$

where the notation $\overline{z}^{p_i}$ means:

$$\overline{z}^{p_i} = \left[z^t \ \frac{dz^t}{dt} \ \cdots \ \frac{d^{p_i} z^t}{dt^{p_i}} \right]^t$$

The matrices $OBS(C_i, A_i, p_i)$ and $COM(A_i, B_i, C_i, D_i, p_i)$ are the observability and control matrices of order p_i for model i.

Defining the matrix W_i such that:

$$W_i[OBS(C_i, A_i, p_i)\ COM(A_i, G_i^1, C_i, G_i^2, p_i)] = 0$$

it is possible to to obtain the analytical redundancy relations as:

$$W_i\overline{y}_i^{p_i} - W_iCOM(A_i, B_i, C_i, D_i, p_i)\overline{u}_i^{p_i} - W_iCOM(A_i, F_i^1, C_i, F_i^2, p_i)\overline{\phi}_i^{p_i} = 0.$$

The computational expression of the structured parity residuals r_i at order p_i is:

$$r_i(t) = W_i\overline{y}_i^{p_i} - W_iCOM(A_i, B_i, C_i, D_i, p_i)\overline{u}_i^{p_i}$$

while the evaluation expression is:

$$r_i(t) = W_iCOM(A_i, F_i^1, C_i, F_i^2, p_i)\overline{\phi}_i^{p_i}.$$

In real problems the residuals are computed in discrete time, i.e. at each time $t = kT_e$ where T_e is the sampling period. Usually residuals are never exactly zero, due to some noise. Then decision procedures must be adopted to decide if the fault has occurred or not. A typical procedure consists in computing the main value of the residual on a sliding time window and then fix a threshold.

The extension of this approach to hybrid systems is based on the definition of discernibility between modes (see [20] for the definition). In this context parity residuals can be used as indexes of consistency, meaning that the non-discernibility between two modes can be checked using the residuals for the two modes: if they both are equal to zero when the same input/output pair is applied then the two modes are said to be weakly non-discernible (where the weakness is due to the fact that the considered model is the nominal one). Moreover this property is also related to the ranks of the observability and control matrices above defined.

Finally, under the assumption that the discrete dynamics of the hybrid system are slow (time interval between two transitions larger than the parity space order p), and that all modes are discernible, it is always possible to compute the actual mode using the residuals, by checking which ones are different from zero in the mean sense.

If the initial mode of the system is unknown, however, it can be determined only by computation of all residuals for all modes in parallel. If then there is some noise the computation of the mean values will lead to delay in the identification of the mode as big as the time window used for the mean value. This can lead to a big computational burden and to a delay in the fault identification.

If the initial/current mode is known, only the residuals of the possible mode successors have to be computed.

Using this approach it is possible to identify the actual mode, to detect and isolate faults in the actual mode and to detect the switching instants to determine the successor mode. These last features allow to detect faults affecting the discrete evolution, because the results of the switching detection can be compared to the previewed nominal one. To detect the switching times a procedure based on the residual mean values computation is used (see [21] for details).

Obviously when two modes are not discernible, it is impossible to detect faults on the discrete evolution. This situation can be avoided using more sensors.

4.2.3 Comparison

Both the above recalled methods use hybrid automata as a modelling framework. These models are very powerful in terms of graphical representation and impact on operators. Moreover they

are able to represent all possible physical configurations of the system, which means that they can be used in diagnosis to find out which configuration is faulty or directly which fault has occurred. Due to the fact that they represent all sequences of states the system can execute, however, hybrid automata usually are subject to state explosion, and the number of states can increase with the complexity of the system itself.

The first approach is quite powerful in detecting many kinds of faults. This is due to the fact that the discrete and the continuous dynamics of the system are diagnosed separately. This means that the most part of the faults (either affecting discrete or continuous dynamics) can be detected using the above presented diagnoser. Moreover there is a sort of intelligent part of the diagnoser, which can also be called supervisor, able to isolate the fault, i.e. to locate it. The major drawback of this method is the big amount of notation which has to be introduced to be able to explain what is going on in the system. Indeed, despite the effort of the authors, some points are still obscure and not well stated. Among them it is interesting to notice that in the introduction and modelling section of [27] the authors claim they represent each component of the system as a hybrid automaton to compose them in a second time. But in the following they never explain how this is useful, because they only present detection for a part of the system. Then it is impossible to understand if the diagnostic procedure is able to accomplish its task in a distributed way or in a centralised way, and in both case which specifications are needed. The representation of the diagnoser as a hybrid system is quite new and useful because it can be composed with the other automata. Nevertheless, due to the lack of clarity on the decomposition features of the procedure, it is not possible to define an integration policy with the diagnostic results on the other parts of the system.

If applied in a centralised way this first approach suffers, as all the automata-based approaches, of the state explosion problem. However the decentralised way seems quite interesting for distributed hybrid systems, because it is able to detect faults both in the continuous and the discrete part, and the discrete faults here considered are events. For this reason interesting extensions to this method are related to the distributed diagnosis problem.

The second approach uses the common parity space theory and tries to extend the results to hybrid systems. In particular the authors try to adapt the model of hybrid systems and faults in hybrid systems to the classical parity space approach. The systems are modelled using hybrid automata, where the transitions are given by switching surfaces. This means that the faults are not really viewed as events, but their effect on the system is taken into account. In this sense the presented method is able to detect faults when they change the continuous dynamics inside each mode. The only kind of discrete faults this approach is able to detect are the ones which bring the system in one mode different from one of the expected successors of the the actual mode. The authors do not go deep into detail in the way they used to generated residuals able to cope with the hybrid nature of the system. Moreover they say that in case the initial mode is unknown the residuals of all the modes of the system must be computer in parallel, leading to a great computational burden, which increases with the complexity of the system.

4.3 State Estimation in Hybrid Systems

As mentioned in previous sections, the knowledge of the operating mode of the system leads to the knowledge of its behaviour and as a consequence to the possibility to establish if the system is still working in nominal conditions or in some unexpected conditions. Hence in this section some estimation strategies for hybrid systems are reported.

State estimation for hybrid systems is a very complex task, especially if autonomous mode transitions are taken into account (that would lead to a more general class of hybrid systems).

In this case the system dynamics are even more coupled, thus more difficult to estimate. For this reason, indeed, the discrete mode and the continuous state have to be estimated in a coupled way, and it is impossible to use methods that state all their combinations because of the high computational burden.

The general state estimation problem consists in determining an estimation of the state $x(T)$ of a system, given the model structure and a sequence of noisy observations/measurements of the system $Y(T) := y(0), ..., y(T)$. In the stochastic framework the state estimation problem consists in finding the conditional density of the state given the measurements, $p(x(T)|Y(T))$. Some interesting ways of solving this problem for systems with continuous dynamics only is using Particle Filtering or Moving Horizon Estimation (see [62] and references therein).

The Particle Filtering (PF) estimation method is an improvement of the Markov Chain Monte Carlo methods, because it provides a recursive approach to probability distribution estimation. To the samples are given some weights generated recursively by the procedure. This usually creates some degeneracy of the weights, i.e. the recursive update of the weights can increase their variance due to particles with small weights. This phenomenon can be dealt by resampling the particles to remove the ones with very small weights. Another problem of PF methods is their strong dependence on the initial guess.

Moving Horizon Estimation (MHE) is based on least squares methods. It is appealing because of its capability of incorporating nonlinearities and constraints on states and disturbances. Although this kind of algorithms amount to optimization problems of finite dimension without the need of all the data (full estimation problem) before the estimation time, the MHE strategy could lead to high real time computational burden for complex system, especially in presence of a huge number of modes.

Some extensions of these techniques to state estimate of hybrid systems have been presented in [44] for PF and in [25] for MHE.

In [25] hybrid systems are considered in PWA form:

$$\begin{array}{rcl} x(t+1) & = & A_i x(t) + f_i + w(t) \\ y(t) & = & C_i x(t) + g_i + v(t), \text{for} x(t) \in \mathcal{X}_i \\ x & \in & \mathbb{X} \\ w & \in & \mathbb{W} \end{array} \tag{4.1}$$

where $v(t) \in \mathbb{R}^{p_c} \times \{0,1\}^{p_l}$, $\mathbb{W} \subset \mathbb{R}^{n_c} \times \{0,1\}^{n_l}$ is a bounded polyhedron containing the origin, w and v model unmeasured system noise and output disturbances respectively. The cost functional used for the optimization problem is

$$J(\tau, t, w, v, x(\tau), \Gamma_\tau) = \sum_{k=\tau}^{t-1} \|v(k)\|_R^2 + \|w(k)\|_Q^2 + \Gamma_\tau(x(\tau))$$

where $\tau, t \in \mathbb{N}, \tau < t, \Gamma_\tau$ is a continuous function and Q and R are positive-definite matrices. The MHE procedure solves at each time instant $T \geq 0$ the problem:

$$\min_{x(T-M),w} J(T-M, T, w, v, x(T-M), \Gamma T-M) \text{subj to 4.1}$$

where $M \in \mathbb{N}^+$ is a fixed time horizon, $T > M$, $\Gamma T - M)$ is a sequence of initial penalties approximating the arrival cost $\overline{\Gamma} T - M)$.

In [44] PF methods are applied to state estimation of distributed hybrid systems. These systems are composed by a set of components modelled as hybrid systems communicating with each other using a network (see Sec. 2.3). The authors are interested in using estimation

algorithms that can be able to cope with this kind of systems where the guard conditions can be nonlinear and the noise can be represented by arbitrary multi-modal distributions. Each hybrid subsystem can be represented by a nonlinear discrete-time model for the continuous dynamics, plus a transition function representing the evolution of the discrete state:

$$\begin{aligned} x(t+1) &= f_q(x(t), u(t)) + w(t) \\ y(t) &= g_q(x(t)) + v(t) \\ q(t+1) = \delta(q(t), \sigma(t), x(t)) & \end{aligned}$$

where $\sigma(t)$ denotes events corresponding to the control commands. In the model presented in this work the coupling between the subsystems occurs through the guards that govern the transitions, this means that a mode transition in a subsystem could be triggered by a condition on the (continuous) state of another subsystem. The state of the whole system is given by all the states of the subsystems. Hence the work is motivated by a computational reason: the centralized estimation uses algorithms computationally very expensive and requires high bandwidth networks, while a distributed estimation can exploit the computation embedded in each component, with obvious computational advantages. The estimation problem here requires the knowledge of the observation sequence $Y(t)$, of the continuous control inputs $U(t)$ and of the history of the control events. Local state estimation requires only the knowledge of the guards transitions that couple the subsystems. Autonomous transitions are monitored using the appropriate mode to update the estimate of the continuous state. The probability of mode transitions due to control commands can be computed using discrete estimation techniques such as the ones based on hidden Markov models. The algorithm presented in [44] evaluates at every time step the transition probability matrix

$$T_{ij}(t) = p(q(t) = j | x(t-1), q(t-1) = i), i, j = 1, \ldots, |Q|$$

where Q is the finite state space of the discrete dynamics. This matrix is evaluated based on the estimate of the continuous state as

$$T_{ij}(t) = \int_{G_{ij}} p(x(t-1)|Y(t-1), U(t-1), q(t-1) = i) dx(t-1)$$

Then the algorithm focuses on the most likely modes and updates the continuous estimate by conditioning on the new measurements based on particle filtering where the computation of transitions probabilities is made through Monte Carlo methods. The transitions probabilities are used to (dynamically) assign particles to the discrete modes (i.e. focusing on the most likely transitions). This algorithm is distributed because the local estimators exchange each other messages with the probability values of the guard conditions defining the coupling. This means that the only part in this algorithm requiring a knowledge about the coupling between the different subsystem is the computation of the mode transition probability. This task can be accomplished locally if at every step the information about the probabilities of local mode transitions due to remote continuous states are sent through the network (see Section **??** for details on distributed hybrid systems).

An advanced estimation algorithm for hybrid systems is presented in [38]. The author here uses some methods from artificial intelligence and model-based reasoning applied to complex systems composed by many components. Each component is considered as an hybrid system and represented with an hybrid automaton. Probabilistic mode transition models are combined with stochastic discrete-time difference equations to obtain a probabilistic hybrid automaton (PHA). The composition of all the PHA of the components gives the concurrent PHA (cPHA)

of the whole system. The estimation techniques used in [38] are sub-optimal methods based on multi-model filtering methods with advanced search and reasoning methods from AI. These choices has been made to handle with complex multi-modal systems with an online strategy. To decrease the computational burden for the estimation task, the author proposes a method based on the decomposition of the system first into its components for a modelling purpose, than following its structural properties. The hybrid estimation problem is reformulated as a best-first search problem. The search techniques used are shortest path and N-step search. The continuous state is estimated through a set of Kalman filters, one for each mode, designed online using a model-based mechanism. The design of the filters is based on a decomposition of the cPHA representing the whole system. This decomposition comes from a causal analysis of the system, through which it is possible to obtain a cause-to-effect chain of variables. Then a structural analysis (see Sec. 3.1) helps in finding the observable parts of the system and to build some filters on sub-parts which are not strongly coupled. This will lead to a bank of filters which should have the property of being reusable, then they are cached as building blocks for filters clusters.

This last approach applies the same tools of the procedure presented in Chapter 3 for distributed systems, extending their use to distributed hybrid systems. For this reason it is quite interesting the idea of exploiting this methodology to obtain an architecture as the one introduced in Section 3.2, where fault diagnosis and reconfiguration are performed in a decentralised way for distributed hybrid system.

4.4 Hybrid Systems Identification Methods

State estimation can be used for fault detection when the parameters of the system are known. When they are unknown system identification is needed. As stated above, the knowledge of the behaviour of the system in each time instant is necessary to detect the occurrence of a fault. This leads to the need of methods able to tell what is the model of the system in a certain time instant and to compare the obtained results with the known effects of the faults in order to give a diagnostic response.

For the sake of completeness of this chapter on the methods for fault detection for hybrid systems, in the following a comparison between two rather different techniques for hybrid systems identification is presented. The main difference is given by the model of the considered systems: in [26] the hybrid systems are given in an input/output form, while in [8] a state-space model is used.

4.4.1 A clustering technique

In [26] a method for identification of discrete-time hybrid systems is presented. The considered systems are represented in a PWA form. The aim of this technique is to identify the system given a certain amount of data. The input-output representation of PWA systems is used, obtaining the so-called PWARX model.

The model of the system is then given by the input-output relation:

$$y(k) = f(x(k)) + \epsilon(k), \tag{4.2}$$

where $f(\cdot)$ is the PWA map defined by:

$$f(x) = \begin{cases} \theta_1' \begin{bmatrix} x \\ 1 \end{bmatrix} & \text{if } x \in \mathcal{X}_1, \\ \vdots & \\ \theta_s' \begin{bmatrix} x \\ 1 \end{bmatrix} & \text{if } x \in \mathcal{X}_s. \end{cases} \tag{4.3}$$

In this model k is the time index and the vector of regressors is:

$$x(k) \triangleq \begin{array}{l} [y(k-1)y(k-2)\dots y(k-n_a) \\ u'(k-1)u'(k-2)\dots u'(k-n_b)]' \end{array} \tag{4.4}$$

where $y \in \mathbb{R}$ and $u \in \mathbb{R}^n$ are the outputs and inputs of the system, and n_a, n_b are the model orders.

The assumptions for solving the identification problem are:

- the number of submodels s is given;
- the orders n_a and n_b are fixed and known;
- as a consequence of this last assumption the regressor set $\mathcal{X}$ is known from the constraints on the inputs and outputs.

The aim of the procedure is to estimate the partition $\mathcal{X}_i, i = 1, \dots, s$ and the parameter vectors $\{\theta_i\}_{i=1}^s$.

The procedure is composed of six steps described in the following.

Step 1 The first step consists in building local data sets (LDs) from the data. This relies on the fact that PWA maps are locally linear, i.e. small subsets of points $x(k)$ closed to each other are supposed to belong to the same region. Then for each datapoint $(x(j), y(j)), j+1, \dots, N$ an LD is built containing $(x(j), y(j))$ and the $c-1$ nearest datapoints. Some LDs will contain only points belonging to the same submodel and will be called pure LDs, whereas the others are called mixed LDs. The procedure is based on the assumption that the ratio between mixed and pure LDs is small and that as the number N of input/output pairs goes to infinity, this ratio goes to 0.

Step 2 A parameter vector representing each LD is identified. This is achieved using least squares estimation to obtain an affine model using the samples in each LD.

Step 3 In this step the parameter vectors are partitioned in s clusters minimizing some functional.

Step 4 This step regards the classification of the original data, obtained using the clustering of the parameter vectors through the bijective maps between these vectors and the $\mathcal{X}$-points.

Step 5 With the information obtained by the previous steps, the s submodels composing the system are estimated. This step can be performed in different ways, either using the weighted least square method (exploiting the results obtained in step 2) or using robust regression techniques.

Step 6 Finally, using a linear classification algorithm, the partition $\mathcal{X}_i, i = 1, \dots, s$ is estimated. This basically means that the separating hyperplanes for each pair of subspaces are searched. These hyperplanes exists because the sets $\mathcal{X}_i$ are polyhedral and convex.

4.4.2 A subspace method

In [8] a new approach to identification of PW linear systems is introduced. This method is based on subspace identification procedure presented in [73]. In particular the approach is based on the switching detection performed using projected subspaces from batches of input/output data.

The considered systems are piecewise linear (PWL) described by the set of state-space equations:

$$\begin{aligned} x(k+1) &= A_i x(k) + B_i u(k) \\ y(k) &= C_i x(k) + D_i u(k) \end{aligned} \tag{4.5}$$

for

$$\begin{bmatrix} x(k) \\ u(k) \end{bmatrix} \in \mathcal{P}_i$$

where $y(k) \in \mathbb{R}^l$ is the output, $u(k) \in \mathbb{R}^m$ is the input and $x(k) \in \mathbb{R}^n$ is the state of the system. $\mathcal{P}_i,\ i = 1, 2, \ldots, M$ is a partition such that:

$$\bigcup_{i=1}^{M} \mathcal{P}_i = \mathbb{R}^n \times \mathbb{R}^m \qquad \text{and} \qquad \mathcal{P}_i \cap \mathcal{P}_j = \emptyset \text{ for } i \neq j.$$

For identification purposes the system 4.5 can also be rewritten as:

$$\begin{aligned} x(k+1) &= \textstyle\sum_{i=1}^{M} p_i(k) \left(A_i x(k) + B_i u(k)\right) \\ y(k) &= \textstyle\sum_{i=1}^{M} p_i(k) \left(C_i x(k) + D_i u(k) + v_i(k)\right) \end{aligned} \tag{4.6}$$

where $v_i(k)$ is zero mean white noise and $p_i(k)$ is the switching signal determining which local model i is active at time k. This signal should satisfy:

$$p_i(k) \in \{0, 1\}, \qquad \sum_{i=1}^{M} p_i(k) = 1. \tag{4.7}$$

The formulation in eq. 4.6 implies hard switchings between the local systems.

The problem of identification of system (4.6) consists in the determination of the switching signal $p_i(k)$ [8] and the estimation of the state-space model [73] from a finite number N of measurements of the inputs and outputs of the system.

Basically the problem of switching detection is solved using the order detection mechanism of subspace identification, but applied to rather smaller batches of data selected using a moving window. The outputs of each batch are projected onto the orthogonal complement of its inputs. If there is no switching in the considered batch the dimension of the projected subspace is equal to the the order of the local system, otherwise the order increases. The method is then based on the rank variation detection of projected subspaces computed from successive batches of data. The data are then classified according to the subspace dimension and basis (using a measure for the distance of the projected subspaces). Finally to the data belonging to the same local system a weight (p_i) is assigned.

This procedure can be used to detect changes when the switching local systems:

- are completely different;
- have different dynamics (poles);
- have different zeros.

The second step is the estimation of the matrices of the system. This is done starting from the knowledge of the switching signals p_i at each time instant k. In [73] the MOESP subspace algorithm is used to achieve this goal up to a linear state transformation. Then a discussion on methods to bring all the systems in the same basis in done. The authors propose an improvement of the comparison of the state of model before each state transition with state of the model after it. The improvement consists in using more than one transition to determine univocally one state transformation.

4.4.3 Comparison and Possible Improvements

In this section some differences and common points between the two approaches are highlighted, while in the next section some simulations will help understanding the main advantages and disadvantages of the each method.

The clustering method is based on the input-output representation of PWA systems, whereas the subspace method is based on the state-space representation of PWL systems. This means that the clustering technique is applicable to a wider number of systems. Moreover the model (4.6) represents a kind of switching systems, where the switches are hard.

Another main difference between the two methods is that the clustering considers single outputs systems, while the subspace considers multi outputs systems. This is an advantage of the subspace method, even if it only takes the batches of data without giving them a physical connotation in first instance. This is indeed due to the fact the subspace methods are used, leading to the necessity of projecting the data to obtain the state of the system. In the clustering method, instead, the data are classified in vectors of regressors, giving to them the connotation of input and output. There is then the need of performing a clustering and classification of data at the beginning of the procedure.

As for the specifications, both methods need the number of submodels, even if in the clustering method it can also be omitted. Then other clustering methods can be exploited and improved to establish the number of submodels. Another hidden specification for the subspace method is the shape of the subspaces in which the submodels evolve.

Finally both methods can be used to detect any kind of switching. In other words the methods are able to cope with submodels having different structure (order, dynamics, etc.). Anyway the assumptions for the application of the methods should always be valid, i.e. the submodels orders need to be known.

An interesting improvement, with the aim of fault tolerance, is to extend the subspace procedure to piecewise affine systems. The class of PWA systems is able to model more situations than the class of piecewise linear systems. Moreover for fault tolerance the use of state-space models allows the modelling of more faults, and the possibility to compare faulty models, representing the effects of the faults on the system, with the actual model directly computed by the subspace identification algorithm. However the extension of the presented results to PWA systems requires a modification of the projection interpretation, with a consequent modification of the switch detection algorithm.

4.5 Detecting Faulty Connections in Hybrid Systems Networks

Communication faults can be intended in two different ways depending on the considered system. If the system is not distributed than the communication can be from a computer where the software is running and the hardware system. In this case a communication fault will affect the estimation/control algorithm and lead to a loss of information for the whole system. If the system

is distributed a communication fault can also be thought as a fault in the network connecting the nodes of the system. This kind of fault is really dangerous, because it can propagate its effect throughout the nodes leading to a real loss of performance. This means that a path to a node is lost and that node will not receive the information from the other nodes unless a new path is found. For this reason it is very important to detect this kind of fault as soon as possible, not to lose to much information and to define an error containment region.

The network configuration covers many applications in modern systems. The problem of communication breakdown has to be faced in these situations. Some practical examples are in distributed computing, when many computers collects data from remote servers and they are connected by physical cables subject to vibrations provoking disconnection. Another possible application is in aircraft formations and fleets, for collision avoidance, where the loss of communication can lead to a change of the route and a possible collision between two aircraft.

4.5.1 Networks of Hybrid Systems

Networks of hybrid systems are composed of hybrid systems cooperating through a communication network. The systems in the network can be components of the same hybrid system, but can also be complete hybrid systems communicating through the channels of the network. The subsystems composing the network are called **nodes**.

There are many different configurations for the networks, depending on how the systems are connected. The choice of the configuration affects the whole system complexity which is primarily due to the different ways the systems are connected. Some networks configurations and many other issues on computer networks are presented in [72]. Two main network configurations, depending on the transmission technology, can be found: broadcast networks and point-to-point networks. The broadcast networks use a single communication channel, like for example a bus, shared by all the nodes. In this communication policy the nodes send messages, i.e. signals, also called packets, which are received by all the other nodes. Inside the message there is an address referring to the receiver node. When a node receives a message it checks if the address corresponds to its address and if so it processes the message, otherwise it is ignored. An example of this kind of network is represented in fig. 4.2.

The other configuration is the point-to-point network. In this kind of networks the nodes are connected in pairs. This means that each node can be connected to all the other nodes pairwise. In this section the hybrid systems composing the network are connected point-to-point (see fig. 4.3(a)), meaning also that each system is directly connected with all the others. The choice of this configuration is motivated by its usefulness for fault detection and reconfiguration. Indeed, if a connection is lost, it will be possible to reach a node following a path which is different from the direct one. Another more general reason for this choice is that it is the more widely used configuration for large scale systems, while using, e.g., a broadcast model will increase the transmission time with the scale of the network (see [72] for details). Suppose, then, that the nodes communicate using a protocol similar to the so called *start-stop*. In this protocol the channel can transmit one message at a time in both directions, but not at the same time. The policy of the protocol is the following:

1. the sender sends a *request to send* (req) signal to the receiver;

2. the receiver sends a *receive ready* (rr) signal to the sender;

3. the sender transmits the message;

4. the receiver sends an *acknowledgement* (*ack*) signal to say it has received the message correctly;

5. the sender sends a *clear to send* (*cs*) signal to say it has finished transmitting messages;

6. both sender and receiver disconnect.

For the sake of simplicity in the following it will be considered that the systems always send only one message at a time.

Due to the fact that each node is connected to all the other nodes and then it can send messages to any other node in the network, it is necessary to establish an address for each node. In this way it is possible to recognize the connection which is actually busy in transmitting data and of course the node to which the data are sent. The addresses can be local or global, depending on the network, if some interfaces are used then they will be local, otherwise some global addresses should be used. Assume the use of local addresses: the nodes can be numbered 1 to n and in our representation we can think they are contained in some signals (*ad*) the sender node uses to activate the right channel.

An automaton representing a connection is reported in fig. 4.3(b) where it is shown when the connection is considered busy or idle. The connection is normally in state **Idle** (I). The sender specifies the address of the receiver in the signal *ad* which is related and recognized by the link connecting the correct nodes. Then the connection moves to state **Ready** (R). When the *req* signal is sent by the sender, the connection moves to state **Busy** (B). With the *rr* and *ack* signals the connection remains in B. When the signal *cs* occurs the connection moves back to I.

4.5.2 A modelling framework: qualitative representations

A modelling framework which is of particular interest for fault detection is based on the Artificial Intelligence theory of Qualitative Reasoning. This approach aims at understanding the human common sense, i.e. to catch the interactions between physical phenomena, without knowing the quantitative aspects. Qualitative models are abstractions of the system behavior, they represent qualitative relations among physical systems. Building qualitative models does not require a mathematical knowledge of the systems and their behavior. This kind of modelling is frequently used for fault diagnosis, when we can be aware of the effects of the faults but not of their quantitative description.

Henced, in this work, the nodes of the network are modelled using the qualitative approach presented in [48], based on the quantisation of hybrid systems. A quantised system is a system in which all the continuous dynamics are converted into discrete dynamics. The approach is based on the quantiser which converts a real-valued signal into a sequence of symbols. This means that in a quantised system only symbolic input/output information is available. The behavior of the quantised system is then qualitative and is given by the set of all I/O pairs consistent with the system dynamics and the signal quantisation. Due to this qualitative representation given by the quantiser, its behavior is qualitative and non-deterministic, i.e. it is impossible to predict the qualitative output sequence of the quantised system unambiguously for given qualitative initial state and input. However it is always possible to know with which probability a qualitative output value $[y(k)]$ occurs (here $[y(k)]$ is the quantised output, and k is the time instant).

Using the theory of stochastic automata (see [5]), it is possible to represent the quantised system with a stochastic automaton

$$S(N_x, N_u, N_y, L, P(z(0)))$$

where N_x, N_u, N_y are the sets of states, inputs and outputs of the system respectively, L is the behavioural relation and $P(z(0))$ is the probability distribution of the initial state $z(0)$ of the automaton. The behavioural relation is defined as

$$L(z', w|z, v) = P([x(1)] = z', [y(0)] = w|[x(0)] = z, [u(0)] = v)$$

where the notation $[\cdot]$ refers to quantised signals, z' is the new (meaning next at time $k+1$) state of the automaton, w is the actual output, z the actual state and v the actual input. So the behavioural relation express the probability of the next state to be z' and the actual output to be w if the actual state is z and the actual input is v.

The task of the observation of quantised systems is to find the current qualitative state $[x(k)]$ for the measured sequences of input/output values. This problem is solved by looking for qualitative internal states $[x(k)]$ for which the given I/O pair may occur, this means that the solution will not be unique, but a set of states. In particular when a fault occurs the observation problem becomes a diagnostic problem. It is worth noting that in general faults are quantised phenomena. Then the diagnostic problem is solved by searching for all faults $f(k)$ for which the given I/O pair can occur. Even in this case the result will be a set of possible faults. Both observation and diagnostic methods are based on a consistency check for a given I/O pair and the model. This means that the model must represent all I/O pairs that may occur for the given quantised system, i.e. the model must be **complete**. The author models the fault in a quantised system as the output of the stochastic automaton $S_f(N_f, G_f, Prob(f(0)))$. This means that the model of the quantised system subject to a fault will be a stochastic automaton given by the combination of the stochastic automaton representing the quantised system and the stochastic automaton representing the fault. In [5] state observation and diagnosis of quantised systems are performed using standard methods for stochastic automata.

In [55] the stochastic automata networks are introduced by extending the concept of stochastic automata with the definition of exogenous signals such as inputs, outputs and faults. In this way different subsystems represented by stochastic automata are connected through the external signals, i.e. some outputs of a subsystem can be inputs for another one. The authors make the assumption that the internal coupling signals are not measurable, which lead to some problems when performing decentralized diagnosis, because not all the subsystems are detectable, due to the unknown input/output signals. In the following this assumption will be removed and all the signals will be supposed to be measurable.

In [63] a way to cast distributed quantised systems into stochastic automata networks is presented. Starting from a quantised system as a whole the network of quantised systems is obtained by following three steps:

- the coupling signals among subsystems are quantised;
- the obtained system is manipulated to obtain causal relations (input to output) among the different subsystems;
- each subsystem is transformed in a quantised system obtaining the network.

Attention must be paid to the sampling time of the quantised system, because the sampled system is usually different from the original continuous one, i.e. more coupling signals can be found. This is very important when we try to decompose the system. For this reason the system is decomposed in quantised subsystems after the discretization.

The above presented modelling framework is adapted to the kind of systems we want to study. Each node is represented by a quantised system, modelled by a stochastic automaton.

The connections are represented by systems themselves, as shown in fig. 4.3(b). Connections and nodes exchange data using their input/output signals which are supposed to be known. The signals of the protocol are divided into inputs and outputs as follows:

$$\text{Sender} \begin{cases} req, cs, ad \in N_y \\ rr, ack \in N_u \end{cases} ; \text{ Receiver} \begin{cases} req, cs, ad \in N_u \\ rr, ack \in N_y \end{cases} .$$

4.5.3 Faults modelling and Fault Detection Issues

There are many different kinds of faults that can occur in a system: sensor, actuator, communication faults. The effects of these kinds of faults show up in different parts of the system. Sensor faults change the output dynamics, while the actuator faults change the state dynamics.

In this section a special attention to communication faults is given. They affect the communication channel. Depending on the severity of the fault the channel can be unreliable or completely lost. Unreliable channels are usually handled by the protocol policy, i.e. if some messages are lost or some bits are changed, the protocol usually is robust enough to recall the message or to recover the wrong transmission. The above specified communication policy is weak in this sense, but some extensions exist such as the so called *go back to n* protocol (see [72]). In this protocol the sender has buffers of the sent and received messages because the receiver sends an acknowledgement signal each time it receives a message; if a message is lost the buffers allow to compare the sent and the received messages and go back to the (n^{th}) message lost.

From this consideration it follows that more severe faults, such as the complete loss of a channel, are interesting for the above specified problem, because they cannot be managed by a robust protocol. The model of the connection with this fault is represented in fig. 4.4. Here the fault is called f and after its occurrence the connection moves to the faulty state F. The fault obviously depends on the transmission signals. Indeed, when one of these signals is expected but not received, a fault has possibly occurred.

Note that faults on the nodes are not considered at this level, but a signal missing can also be due to a node internal fault, which should be distinguished from the connection fault. Note also that from the connection point of view it is not possible to relate the fault to one of the signal missing, because the connection is not aware of the nodes communication policy, while the protocol is managed by the nodes themselves.

The connection faults for the presented systems can be detected using a decentralized policy. This allows the fault detection and isolation (FDI) unit not to check all the connections and the nodes even for big complex systems. This can be realised giving a state estimator to each node and a fault detection unit to each pair of nodes. The state estimator aims at giving an estimate of the state of the system to avoid mistaking internal node faults for connection faults, while the fault detection unit supervises the pair of nodes that communicate to detect if a fault on the channel occurs. Practically the estimator checks if the state of the node is the nominal one or one of the possible faulty ones. If the estimators of both communicating nodes claim that they are in the nominal state but the fault detection unit claims that some transmission signal (related to the fault f) is missing, then the fault detection unit can establish that a fault on the connection has occurred.

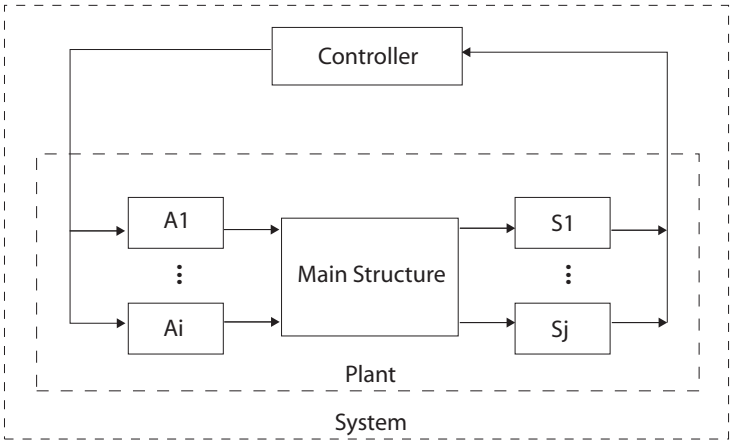


Figure 4.1: Dynamic system.

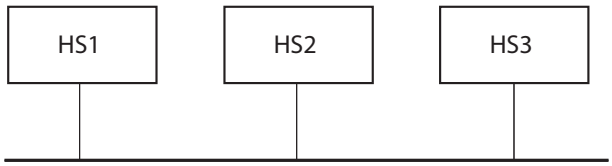
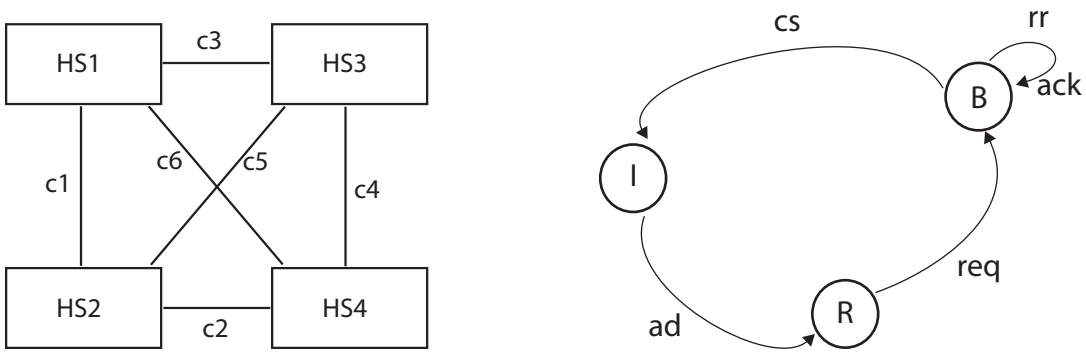


Figure 4.2: Example of broadcast hybrid system network.



(a) Point-to-point HSN.

(b) Automaton of a connection.

Figure 4.3: Hybrid Systems Network (HSN).

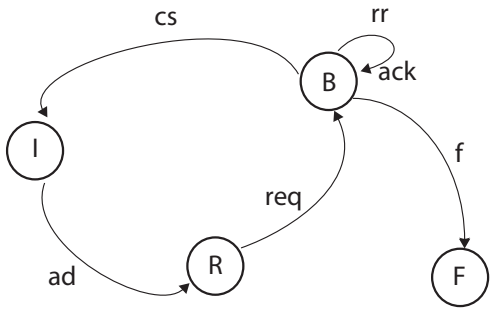


Figure 4.4: Automaton of a connection with the faulty state.

Conclusions

The aim of this three years of thesis was to introduce new methods for dealing with fault tolerance in complex, distributed systems.

The complex modern systems are usually distributed, i.e. they are composed by subsystems connected through a network. These subsystems achieve different functions and communicate to reach a common goal. The subsystems composing a distributed systems can be computers or dynamical systems. Depending on their nature and on the aim of their analysis, they can be represented by discrete event or hybrid systems. A discrete event representation is more used for a higher level description, when only the discrete dynamics and the communication policy is considered. The hybrid representation involves also the continuous dynamics of each subsystem and the interaction between the continuous and the discrete dynamics. Indeed discrete event systems are discrete-state, event-driven systems, while hybrid systems are systems in which continuous and discrete dynamics interact.

Fault tolerance in these large scale systems is quite complex, because it has to consider problems of propagation of the fault effect and problems of scaling. Moreover to achieve fault tolerance without increasing the computational burden some decentralized methods have to be used. These methods can cope with the distributed nature of the systems, with the final aim to decentralize the tasks of fault diagnosis and control redesign.

The main contribution of this thesis is given by the implementation of a procedure to design a fault tolerant architecture for distributed systems. This architecture follows the nature of the considered systems, hence it is modular, hierarchical and fault detection is achieved in a decentralized way. The decentralization is based on functional and structural criteria.

The first analysis to design the architecture is performed at the higher level, without going into the detail of the nature of the subsystems composing the distributed system. A deeper study is presented in the last chapter of this thesis, where some results and new issues for fault detection of hybrid systems are introduced.

A possible extension to the presented results is given by the study of method for achieving fault tolerance in distributed hybrid systems, following a decentralised policy. Some applications for the presented results are also an issue.

Curriculum vitae

Marta Capiluppi was born in Bologna in July, 3rd 1978.

In July, 23rd 2003 she took her laurea degree in Computer Science Engineering at the University of Bologna with a thesis titled *Architetture di Controllo Fault Tolerant per Sistemi Distribuiti* (in Italian). In November 2003 she took the professional degree.

From January 2004 to December 2006 she was a PhD student in Control System Engineering and Operational Research within CASY - D.E.I.S. under the supervision of Prof. Claudio Bonivento. The work of the thesis was based on the study of fault tolerant methods for distributed and hybrid systems. From the beginning of her thesis to the end of the project in March 2005 she worked in the framework of European Project IFATIS (Intelligent Fault Tolerant Control in Integrated Systems). From January 2006 to March 2007 she was scientific guest at the Automatic Control Laboratory, ETH Zurich, under the supervision of Prof. Manfred Morari. She is a member of the center CASY (Center for Research on Complex Automated Systems).

Her actual research interests focus on hybrid and networked systems.

Bibliography

[1] J.D. Andrews and T.R. Moss. *Reliability and Risk Assessment*. Professional Engineering Publishing, 2002.

[2] P. Antsaklis and A. Nerode. Guest editorial hybrid control systems: An introductory discussion to the special issue. IEEE Transactions on Automatic Control, 1998.

[3] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:407, 1999.

[4] A. Birolini. *Reliability Engineering: Theory and Practice*. Springer-Verlag, 1999.

[5] M. Blanke, M. Kinnaert, M. Staroswiecki, and J. Lunze. *Diagnosis and fault-tolerant control*. Springer-Verlag, 2003.

[6] C. Bonivento, M. Capiluppi, L. Marconi, and A. Paoli. An integrated design approach to multilevel fault tolerant control of distributed systems. *In proceedings of XVI IFAC World Congress, Prague, Czech Republic*, 2005.

[7] C. Bonivento, M. Capiluppi, L. Marconi, A. Paoli, and C. Rossi. Reliability and safety evaluation for fault diagnosis in distributed systemss. *In proceedings of 6th IFAC Safeprocess, Beijing, PR China*, 2006.

[8] J. Borges, V. Verdult, M. Verhaegen, and M.A. Botto. A switching detection method based on projected subspace classification. In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, Seville, Spain, Dec 2005.

[9] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control. In *Proceedings of the 33rd Conference on Decision and Control*, Lake Buena Vista, Florida, Dec 1994.

[10] L. Marconi A. Paoli C. Bonivento, M. Capiluppi. System analysis and decomposition methods. *IFATIS deliverable D6-3*, 2003.

[11] L. Marconi A. Paoli C. Bonivento, M. Capiluppi. Distributed fault tolerant control of a two-tanks system. *In proceedings of ACD Workshop 2004n*, 2004.

[12] L. Marconi A. Paoli C. Bonivento, M. Capiluppi. An integrated design approach to multilevel fault tolerant control of distributed systems. *In proceedings of ACD Workshop 2004n*, 2004.

[13] L. Marconi A. Paoli C. Bonivento, M. Capiluppi. Report on the integrated design approach for the multilevel ftc system. *IFATIS deliverable D6-4*, 2004.

[14] M Capiluppi. Functional analysis of distributed systems using structural graphs. Poster session, Special CASY Workshop on Advances in Control Theory and Applications, 2006.

[15] M. Capiluppi and A. Paoli. Distributed fault tolerant control of the two-tanks system benchmark. *In proceedings of 44th IEEE CDC and ECC'05, Seville, Spain*, 2005.

[16] M. Capiluppi and M. Staroswiecki. From structural to functional models of complex systems. *In proceedings of 6th IFAC Safeprocess, Beijing, PR China*, 2006.

[17] P. Capiluppi. Large scale computing. Proceedings of the 7th School of Non-Accelerator Astroparticle Physics, Trieste, Italy, 26 Jul - 6 Aug 2004, pages 262-276, World Scientific 2005 Singapore, 2004. ISBN 978-981-256-316-3.

[18] C.G. Cassandras and S. Lafortune. *Introduction to discrete event systems.* Kluwer Academic Publisher, 1999.

[19] E. Cinquemani, M. Micheli, and G. Picci. Fault detection in a class of stochastic hybrid systems. *43rd IEEE Conference on Decision and Control*, 2004.

[20] V. Cocquempot, T. El Mezyani, and M. Staroswiecki. Fault detection and isolation for hybrid systems using structured parity residuals. In *Proceedings of the 5^{th} Asian Control Conference*, Melbourne, Australia, 2004.

[21] V. Cocquempot, M. Staroswiecki, and T. El Mezyani. Switching time estimation and fault detection for hybrid systems using structured parity residuals. In *Proceedings of the 5^{th} IFAC Safeprocess*, Washington DC, USA, 2003.

[22] F. Cristian. Understanding fault-tolerant distributed systems. *Communication of the* ACM, 34(2), 1991.

[23] R. Diestel. *Graph Theory.* Springer-Verlag, 2005.

[24] J. Dowling, R. Cunningham, E. Curran, and V. Cahill. Component and system-wide self-* properties in decentralised distributed systems. Technical report, University of Dublin, Trinity College, 2004.

[25] G. Ferrari-Trecate, D. Mignone, and M. Morari. Moving horizon estimation for hybrid systems. *IEEE Transaction on Automatic Control*, 47(10), 2002.

[26] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39:205–217, 2003.

[27] G.K. Fourlas, K.J. Kyriakopoulos, and N.J. Krikelis. A framework for fault detection of hybrid systems. In *Proceedings of the IEEE MED 2001 Conference*, Dubrovnik, Croatia, 2001.

[28] G.K. Fourlas, K.J. Kyriakopoulos, and N.J. Krikelis. Diagnosability of hybrid systems. In *Proceedings of the IEEE MED 2002 Conference*, Lisbon, Portugal, 2002.

[29] G.K. Fourlas, K.J. Kyriakopoulos, and N.J. Krikelis. Model based fault diagnosis of hybrid systems based on hybrid structure hypothesis testing. In *Proceedings of the IEEE MED 2003 Conference*, Rhodes, Greece, 2003.

[30] P.M. Frank. Fault diagnosis in dynamic systems using analitical and knowledge-based redundancy: a survey and some new results. *Automatica*, 26(3), 1990.

[31] A. L. Gehin, M. Staroswiecki, and M. Assas. Structural analysis of system reconfigurability. *IFAC Safeprocess 2000*, 2000.

[32] J.J. Gertler and D. Singer. A new structural framework for parity space equation based failure detection and isolation. *Automatica*, 26:381–388, 1990.

[33] F. Guenab, C. Join, J.C. Ponsart, D. Sauter, D. Theilliol, and P. Weber. A reliability approach to reconfiguration strategy: application to the ifatis benchmark problem. *In proceedings of IFAC-SSSC04*, 2004.

[34] F. Guenab, D. Theilliol, P. Weber, J.C. Ponsart, and D. Sauter. Fault-tolerant control design based on cost and reliability analysis. *In proceedings of ACD Workshop*, 2004.

[35] F. Hamelin, H. Jamouli, and D. Sauter. The two tanks pilot plant. IFATIS report IFAN014R01, February 2004. http://ifatis.uni-duisburg.de/.

[36] W. P. M. H. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37:1085–1091, 2001.

[37] D.M. Himmelblau. *Fault detection and diagnosis in chemical and petrochemical processes*. Chemical engineering monographs. Elsevier Scientific Pub., 1978.

[38] M. W. Hofbaur. *Hybrid Estimation of Complex Systems*. Lecture Notes in Control and Information Sciences. Springer, 2005.

[39] R. Isermann. Process fault detection based on modeling and estimation methods - a survey. *Automatica*, 20(4), 1984.

[40] B. Jiang, M. Staroswiecki, and V. Cocquempot. Fault accommodation for nonlinear dynamic systems. *IEEE Transactions on Automatic Control*, 51(9), 2006.

[41] K. H. Johansson, J. Lygeros, and S. Sastry. Modeling of hybrid systems. UNESCO Encyclopedia of Life Support Systems, EOLSS, 2001.

[42] C.N. Jones and J.M. Maciejowski. Fault tolerant flight control - an overview. Cambridge University Engineering Department, Technical Report, November 2005.

[43] H. Kopetz. *Real-time Systems: Design Principles for Distributed Embedded Applications*. Real-time systems. Kluwer Academic Publishers, London, 1997.

[44] X. Koutsoukos, J. Kurien, and F. Zhao. Estmation of distributed hybrid systems using particle filtering methods. HSCC 2003, Vol. 2623, in LNCS, Springer, 2003.

[45] U. Lerner, R. Parr, D. Koller, and G. Biswas. Bayesian fault detection and diagnosis in dynamic systems. *7th National Conference on Artificial Intelligence*, 2000.

[46] R.W. Lewis. *Modelling control systems using IEC 61499. Applying function blocks to distributed systems*. The Institution of Electrical Engineers Publishing, London, 2001.

[47] J. Lunze. Diagnosis of discretely controlled continuous systems. *Automatisierungstechnik*, 54(8), 2006.

[48] J. Lunze and J. Raisch. Discrete models for hybrid systems. in Modelling, Analysis and Design of Hybrid Systems, LNCIS 279, Springer, 2002.

[49] J. Lunze and T. Steffen. Hybrid reconfigurable control. In Modelling, Analysis and Design of Hybrid System, editors S. Engell, G. Frehse and E. Schnieder, LNCIS 279, Springer-Verlag, 2002.

[50] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185:105, 2003.

[51] S. McIlraith, G. Biswas, D. Clancy, and V. Gupta. Towards diagnosing hybrid systems. In *Working Notes of the AAAI 1999 Spring Symposium Series: Hybrid Systems and Artificial Intelligence*, Stanford, California, 1999.

[52] D. Mignone, A. Bemporad, and M. Morari. Moving horizon estimation for hybrid systems and fault detection. In *Proceedings of the American Control Conference*, pages 2471–2475, San Diego, California, Jun 1999.

[53] D. Mignone and A. Bemporad M. Morari. A framework for control, fault detection, state estimation, and verification of hybrid systems. In *Proceedings of the American Control Conference*, pages 134–138, San Diego, California, Jun 1999.

[54] S. Mullender. *Distributed Systems*. Addison Wesley Publishing Company, ii edition, 1993.

[55] J. Neidig and J. Lunze. Decentralised diagnosis of automata networks. In *Proceedings of the 16th IFAC World Congress*, Prague, CZ, Jul 2005.

[56] R.J. Patton. Fault-tolerant control: The 1997 situation. In *Proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, Kingston Upon Hull, U.K, 1997.

[57] R.J. Patton, P.M. Frank, and R.N. Clark. *Issues of Fault Diagnosis for Dinamical Systems*. Springer-Verlag, 2000.

[58] L.F. Pau. *Failure Diagnosis and Performance Monitoring*. Marcel Dekker, 1981.

[59] Y. Pencolé and M.O. Cordier. A formal framework for decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164(1-2), 2005.

[60] I.R. Petersen and D.C McFarlane. A methodology for robust fault detection in dynamic systems. *Control Engineering Practice*, 12:123–138, 2004.

[61] B. Randell. Fault tolerance in decentralized systems. *4th Internationa Symposium on Autonomous Decentralized Systems. Integration of Heterogeneous Systems*, 1999.

[62] J. B. Rawlings and B. R. Bakshi. Particle filtering and moving horizon estimation. In *Chemical Process Control, CPC7*, Lake Louise, Alberta, Canada, Jan 2006.

[63] J. Schröder. *Modelling, State Observation and Diagnosis of Quantised Systems*. Lecture Notes in Control and Information Sciences. Springer, 2003.

[64] M. Staroswiecki. Actuator faults and the linear quadratic control problem. *In proceedings of 42nd IEEE Conference on Decision and Control, year = 2003, address = Maui, Hawaii, USA,*.

[65] M. Staroswiecki. Progressive accommodation of actuator faults in the linear quadratic control problem. *In proceedings of 43rd IEEE Conference on Decision and Control, year = 2004, address = Atlantis, Paradise Island, Bahamas,.*

[66] M. Staroswiecki. Fault tolerance systems. In Fault Diagnosis and Fault Tolerant Control, editors P. M. Frank and M. Blanke, Encyclopedia of Life Support Systems (EOLSS). Eolss Publishers, Oxford, UK, 2002. Developed under the auspices of the UNESCO, [http://www.eolss.net].

[67] M. Staroswiecki. Structural analysis for fault detection and isolation and for fault tolerant control. In Fault Diagnosis and Fault Tolerant Control, editors P. M. Frank and M. Blanke, Encyclopedia of Life Support Systems (EOLSS). Eolss Publishers, Oxford, UK, 2002. Developed under the auspices of the UNESCO, [http://www.eolss.net].

[68] M. Staroswiecki. Fault tolerant control : the pseudo-inverse method revisited. *In proceedings of XVI IFAC World Congress*, 2005.

[69] M. Staroswiecki and A. L. Gehin. Analysis of system reconfigurability using generic component models. *UKACC Control'98*, 1998.

[70] M. Staroswiecki and A. L. Gehin. A formal approach to reconfigurability analysis / application to the three tank bechmark. *In Proc. of European Control Conference, ECC'99*, 1999.

[71] M. Staroswiecki, A. L. Gehin, and M. Bayart. Faulty resources management in automation systems. *IEEE SMC'97*, 1997.

[72] A.S. Tanenbaum. *Computer Networks*. Prentice Hall, iv edition, 2003.

[73] V. Verdult and M. Verhaegen. Subspace identification of piecewise linear systems. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, Bahamas, Dec 2004.

[74] Wikipedia. Computer cluster — Wikipedia, the free encyclopedia, 2007.

[75] Wikipedia. Distributed control system — Wikipedia, the free encyclopedia, 2007.