On Fault Handling in Control Systems

Marcel Staroswiecki

Abstract: Whereas fault diagnosis has been the subject of intensive research since the 1970s, the design of fault tolerant systems is a recent research field which does not set its foundations in a unified framework and does not use a unified vocabulary. As a contribution to this special issue, this paper proposes an ontology for the problem of Fault Handling, that embeds the problem of Fault Tolerance.

Keywords: Diagnosis, fault tolerance, fault accommodation, system reconfiguration, fault handling performances.

1. INTRODUCTION

Whereas fault diagnosis has been the subject of intensive research since the 1970s, both in the Control and the Artificial Intelligence communities [1,3,7,9] the design of fault tolerant systems, motivated by many real life applications (aerospace, power systems, safety critical systems), is a recent research field [2,11,17]. Due to this reason, the huge amount of works that can be found in the literature does not set its foundations in a unified framework and does not use a unified vocabulary. As a contribution to this special issue, we propose an ontology for the problem of Fault Handling (FH) which includes - but is not limited to - Fault Tolerance (FT). Quoting [8], "the term ontology [means] a specification of a conceptualization ... Practically, an ontological commitment is an agreement to use a vocabulary (...) in a way that is consistent (...) with respect to the theory specified by an ontology."

The paper builds a framework that allows to evaluate new contributions, to understand in what respect Fault Tolerant Control is different from Control, how Fault Tolerance complements Fault Avoidance, how Fault Handling properties can be ascertained and evaluated for a given design. It is organized as follows : Section 2 defines a (healthy or faulty) system from the interconnected components point of view. The different Fault Handling levels are analyzed in Section 3, and Section 4 illustrates the introduced concepts with the example of an estimation application. Section 5 discusses the evaluation of Fault Handling performances and Section 6 concludes the presentation, by stressing also some implementation problems that still deserve some research.

Manuscript received September 15, 2007; accepted April 21, 2008. Recommended by Guest Editor Bin Jiang.

Marcel Staroswiecki is with SATIE, ENS Cachan, USTL, CNRS, UniverSud 61 avenue du Président Wilson 94235 Cachan Cedex, France (e-mail: marcel.staroswiecki@univ-lille1.fr).

2. SYSTEM DESCRIPTION

A system Σ is a set of interconnected (hardware and software) components. This section presents the component based modeling frame as the natural frame in which fault diagnosis and fault handling concepts can be clearly set.

2.1. Hardware components

According to the selected granularity of variables and time (continuous, discrete, synchronous, asynchronous, etc), different formalisms are used to model the normal behavior of a component $\sigma \in \Sigma$. For conciseness and simplicity, only deterministic systems, continuous variables and continuous time are considered here : the normal open-loop behavior of a hardware component $\sigma \in \Sigma$ is described by state space equations

$$H_p(\sigma): \dot{x}_\sigma - f_\sigma(x_\sigma, v_\sigma, t) = 0, \tag{1}$$

if σ is a process component, or by measurement equations

$$H_m(\sigma): y_\sigma - g_\sigma(x_\sigma, v_\sigma, t) = 0, \tag{2}$$

if σ is a sensor. In (1) and (2), x_σ is a local state, y_σ is an output signal and v_σ is a vector of input signals. Interconnections imply that some variables are common to several components. Therefore, the global state x is the concatenation of all the local states associated with the process components, the global measurement vector y is the concatenation of all the local measurements associated with the sensors, and the global input vector u is the concatenation of all the interconnection variables that are not states.

2.2. Software components

Extending the description to closed loop systems needs equations by which control and estimation signals are synthesized. In a general form, a control (resp. estimation) component σ introduces software constraints $S_c(\sigma)$ (resp. $S_e(\sigma)$)

$$S_c(\sigma) : \varphi_\sigma(\bar{u}, \bar{z}, \bar{y}) = 0, \tag{3}$$

$$S_e(\sigma) : \gamma_\sigma(\bar{u}, \bar{z}, \bar{y}) = 0, \tag{4}$$

where φ_σ and γ_σ follow from the design procedure, z is a vector of estimated variables and for any vector v the notation $\bar{v}$ is defined by $\bar{v}^T \triangleq (v^T, \dot{v}^T, \ddot{v}^T, \ldots)$ (the dimensions of vectors and the exact number of derivatives that are considered need not be specified here).

Note that algorithms $S_c(\sigma)$ and $S_e(\sigma)$ are correctly executed only if other hardware components (computers, memories, communication networks) operate satisfactorily. The behavior, diagnosis and fault tolerance of these components are not considered here, being the subject of an intensive research activity in computer science.

2.3. System behavior

The behavior model of the system is the pair (C,V), where $C \triangleq H \cup S, H$ is the set of all hardware constraints $H_p(\sigma), H_m(\sigma)$, and S is the set of all software constraints $S_c(\sigma)$ or $S_e(\sigma)$ associated with the system components $\sigma \in \Sigma$, and V is the set of all the variables that are involved in these constraints. A subsystem $\Sigma_i \subset \Sigma$ is a subset of components along with their interconnections. Its model (C_i, V_i) is easily defined from the model (C,V).

More generally, a subsystem of Σ is a subset of constraints of C along with the variables they involve. Interconnecting two subsystems whose behavior is (C_1, V_1) and (C_2, V_2) boils down to create the subsystem whose behavior is (C,V) where $C = C_1 \cup C_2$ and $V = V_1 \cup V_2$ (the management of the common variables $V_1 \cap V_2$ is evident and is not detailed here).

2.4. Healthy and faulty situations

No matter whether a component, a subsystem or the whole system is considered, let C and V be the constraints and variables under consideration.

Consistency and falsification: Let $\hat{V}$ be a realization of the variables V. This realization is *consistent* with C if all the constraints are satisfied - this is noted $OK(C,\hat{V})$ - otherwise it is *inconsistent*, i.e., at least one constraint is *falsified* by $\hat{V}$ - this is noted $\rceil OK(C,\hat{V})$. Using the notations $OK(C,V)$: $\forall \hat{V}$, $OK(C,\hat{V})$ and $\rceil OK(C,V) : \exists \hat{V}, \rceil OK(C,\hat{V})$ one has:

Healthy and faulty systems: The pair (C,V) is healthy (noted $\mathscr{H}_0$) if $OK(C,V)$ holds :

$$\mathscr{H}_0 \Leftrightarrow OK(C,V). \tag{5}$$

By negating (5), it follows that the definition of a faulty (C,V) (noted $\mathscr{H}_1$) is :

$$\mathscr{H}_1 \Leftrightarrow \exists c \in C : \rceil OK(C,V). \tag{6}$$

Remark 1: These definitions are independent of any process by which consistency/inconsistency or healthy/faulty conclusions are reached. Indeed, the consistency $OK(C,\hat{V})$ or inconsistency $\rceil OK(C,\hat{V})$ conclusion needs the knowledge of the realization $\hat{V}$, which is available only if the variables are measured or if they are estimated from other measured variables. The $\mathscr{H}_0$ conclusion needs $OK(C,\hat{V})$ to be checked true for all possible realizations of V, which is impossible if there is an infinite number of them. It is a well known fact that an hypothesis e.g., $\mathscr{H}_0$ can always be falsified (it is sufficient to observe an inconsistent realization), but it is impossible to prove that it is always true [4,12]. This is obviously the difficulty of diagnosis algorithms (see below).

Fault modes: A *fault* is defined as the violation of one or several constraints in C (a hardware fault violates constraints in H while a software fault violates constraints in S). Note that this definition does not assume any fault model, and that it covers both the single and multiple fault cases. Indeed, if all the violated constraints belong to the same subset $C(\sigma)$ for some $\sigma \in \Sigma$, then only component σ is faulty, while otherwise several components are faulty. Obviously, there are $\left|2^C\right| - 1$ different subsets of constraints that can be violated, resulting in as many *fault modes*, where 2^E and $|E|$ are respectively the power set and the cardinal of a set E.

System models: The set of constraints that should apply following the system design is the nominal system model, C_n. Let $C_f \neq C_n$ be the constraints that apply under fault mode f, the pair (C_f, V) is the *model of the faulty system*. When needed, different representations can be used for C_f. For hardware constraints, *additive fault models* introduce deviations in the constraints value, for example (1) - (2) is replaced by

$$\dot{x}_\sigma - f_\sigma(x_\sigma, v_\sigma, t) = \psi_\sigma^s(t),$$
$$y_\sigma - g_\sigma(x_\sigma, v_\sigma, t) = \psi_\sigma^m(t),$$

where $t \geq t_f \Rightarrow (\psi_\sigma^s(t), \psi_\sigma^m(t)) \neq (0,0)$ and t_f is the fault occurrence time, while *parametric fault models* introduce deviations in the constraints parameters, for example $\dot{x}_\sigma - f_\sigma(x_\sigma, v_\sigma, \theta, t) = 0$ and $y_\sigma - g_\sigma(x_\sigma, v_\sigma, \theta, t) = 0$ with θ a parameter whose value is θ_n for the nominal system and $\theta_f \neq \theta_n$ for the faulty system. Other models are used for software faults, they are not detailed here since only hardware faults will be considered in the sequel.

Uncertain and stochastic systems: For simplicity, only one set of deterministic constraints C_n was introduced as the *model of the nominal system.* Uncertain systems can also be considered, by generalizing definition (5) to

$$\mathscr{H}_0 \Leftrightarrow \exists C_n \in \mathscr{C}_N : OK(C_n, V),$$

where $\mathscr{C}_N = \{C_n, n \in N\}$ is the set of possible constraints in normal operation. Similarly, stochastic systems can also be considered, by introducing an extra set of random variables, and a description of their distributions in normal (faulty) operation [11,18].

2.5. System objectives

When designing a system, hardware components (i.e., constraints) are selected and interconnected, and software constraints are designed in order to achieve specific objectives.

Objective: An *objective* is a *set of specifications* (or *properties*) P that the system behavior must satisfy. *Functional* and *safety* objectives are distinguished, according to whether the associated specifications concern the normal operation (e.g., exhibit a certain stability degree, exhibit given handling qualities, track some trajectory with given performances, estimate some states with given accuracy) or the incidental / accidental operation (e.g., remain controllable for an emergency stop, a controlled shut-down sequence).

Valid Configurations: For a given objective P to be satisfied, there may be different subsets of hardware and software components that are able to achieve it. Each subset is called a *valid configuration*, and its behavior is described by a specific model. The notation $P(H,S)$ stands for "The subset of components H, controlled with algorithm S satisfies property P, or in other words (H,S) is a valid configuration for objective P.

Degraded Performances: Defining the objective may include specifying required performances. In some cases, it may be useful to define both nominal and degraded performances, the former being achieved in nominal operation, the latter being accepted in faulty operation [10,15].

Weaker property: Degraded performances are defined by specifying a property P^- that is *weaker than* P, i.e., such that

$$\forall (H,S) : P(H,S) \Rightarrow P^-(H,S).$$

Therefore, the pair (P, P^-) describes the performances that are expected in normal operation, together with the level of degradation that can be accepted.

User Selected Operating Mode (USOM): Since control systems are expected to achieve different objectives at different times (for example initialization, production, cleaning, shut-down, etc.), a hybrid description is necessary. A *USOM* is a discrete state in which the system is expected to achieve a given objective. The **current USOM** defines the objective that is requested at a given time. It can be changed (meaning that objective P is replaced by objective P') either following a *spontaneous transition* - for example, the landing mode is changed into the taxying mode as soon as the plane has reached the runway - or by a *controlled transition* - replacing for example the autopilot mode by the manual control mode, or changing the production mode into the shut down mode.

USOM Automaton: The set of USOMs and the set of transitions between them defines an *automaton* which describes the discrete state behavior of the system, i.e., all the objectives the system can be given. Its nodes are the system USOMs and its transitions describe how USOMs are left and entered [5,6].

2.6. Diagnosis algorithms

Diagnosis algorithms are in charge of generating knowledge about the fault. Typically, three problems must be solved at any time t:

Fault detection (FD): From the evolution of the (known) inputs and outputs on the time window $[0,t]$ *find the operating regime*, $\mathscr{H}_0$ or $\mathscr{H}_1$, that is most likely to be true and the time $t_d \leq t$ when it started being true.

Fault isolation (FI): From the evolution of the known variables on the time window $[0,t]$ *find the subset of the constraints* C_n that are falsified, if some violation has been detected (remember that the link between subsets of constraints and components is known).

Fault estimation (FE): From the evolution of the known variables on the time window $[0,t]$ *find the model of the falsified constraints* C_f that describe the behavior of the faulty system.

Remark 2: (i) some faults are **non detectable**, which means that both $\mathscr{H}_0$ or $\mathscr{H}_1$ are consistent with the observed data, (ii) some faults are **non isolable**, which means that several subsets of falsified constraints are consistent with the observed data, (iii) fault estimation needs a class of fault models to be defined, and (iv) each of the above problems can receive imperfect solutions, namely for FD : false alarms, missed detections, detection delay, for FI : inaccurate isolation, mis-isolation, and for FE : wrong model class, wrong model parameters.

3. FAULT HANDLING LEVELS

The fault handling (FH) problem is a decision problem in the presence of a fault. It can be summarized under two points:

1) decide if the current system objective can be achieved in spite of the fault, and control the system accordingly,
2) when the objective cannot be achieved, decide about another objective, and control the system accordingly.

In the first case, the objective is tolerant to that specific fault and the fault is said to be recoverable.

Recoverable Fault: A fault is *recoverable* if the current system objective can be achieved *in spite of its occurrence.*

There are three fault handling levels, that are addressed in this section : recoverable faults can be recovered either by passive or by active fault tolerance (in that case, by using either an accommodation or a reconfiguration strategy), while objective reconfiguration is associated with non recoverable faults.

3.1. Passive fault tolerance

Passive fault tolerance is the first FH level [2]. Let H_f be the faulty hardware and S_n be the nominal software, PFT is associated with the question : "is the configuration (H_f,S_n) valid"? In other terms, when controlled by the nominal algorithm S_n, is the faulty hardware H_f still able to satisfy property P, i.e., do we have $P(H_f,S_n)$ - or $P^-(H_f,S_n)$ if degraded performances are allowed?

Passive Fault Tolerance: Passive Fault Tolerance (PFT) is the strategy by which a fault H_f is recovered using the configuration (H_f,S_n). Fault H_f is *passively tolerated* if $P(H_f,S_n)$ holds true. The system is *passively fault tolerant with respect to a given set of faults* F if $P(H_f,S_n)$ $\forall H_f \in H_F$ where $H_F=\{H_f, f\in F\}$ i.e., any fault in the set is passively tolerated.

Passive Fault Tolerance design: Given a set of faults F with respect to which property P is wished to be tolerant, the passive fault tolerance design problem is therefore

$$\text{Find } S_n \text{ such that} \begin{cases} P(H_n,S_n) \text{ (nominal system)} \\ \forall f \in F: P(H_f,S_n) \text{ (faulty system).} \end{cases}$$

Remark 3: (1) PFT is nothing but robustness since the algorithm S_n is able to achieve the desired property, whatever the system behavior H_n or $H_f \in H_F$. (2) Performance degradation may always be allowed, therefore in the above definitions, $P(H_f,S_n)$ should write: "$P(H_f, S_n)$ or $P^-(H_f,S_n)$ if degraded performances are allowed". This longer formulation is implicit here and everywhere in the sequel. (3) Deciding whether a particular fault f can be passively tolerated or not needs the behavior model H_f to be known.

3.2. Active fault tolerance

Active fault tolerance is the second FH level, it is associated with the question : "is there a configuration (H,S) that is valid i.e., such that $P(H,S)$"? The question admits two variants, according to whether H is constrained to be H_f or may be different.

Fault Accommodation: Fault Accommodation (FA) is the strategy by which a fault H_f is recovered using a specific configuration (H_f,S_f). In other words, a fault H_f is FA-recoverable if there is a software S_f such that $P(H_f,S_f)$, i.e., when controlled by the new algorithm S_f, the faulty hardware H_f is able to satisfy property P.

Fault Accommodation design: Given a set of faults F with respect to which property P is wished to be tolerant, the FA design problem is

$$\forall f \in F \text{ find } S_f : P(H_f,S_f).$$

Remark 4: (1) FA can be of interest in both cases where PFT is possible or not. (2) Deciding whether a particular fault f is FA-recoverable and, if it is, designing the accommodated control, needs the behavior model under this fault, H_f, to be known.

System Reconfiguration: System Reconfiguration (SR) is the strategy by which a fault H_f is recovered using a configuration $(\bar{H}_f,S)$ with $\bar{H}_f \cap H_f = \varnothing$. A fault H_f is SR-recoverable if there is a configuration $(\bar{H}_f,S_f)$ that contains no faulty constraint and which is valid.

System Reconfiguration design: Given a set of faults F with respect to which property P is wished to be tolerant, the SR design problem is

$$\text{For } f \in F \text{ find}(\bar{H}_f, S_f),\ \bar{H}_f \cap H_f = \varnothing : P(\bar{H}_f, S_f)$$

Remark 5: (1) Obviously, SR can be of interest in both cases when FA or PFT apply. (2) SR does not need the model of the faulty hardware H_f to be known, since there is no faulty constraint in $\bar{H}_f$. (3) SR can be formulated in a more restrictive way, namely: "a fault H_f is SR-recoverable if there is a valid configuration $(\bar{H}_f, S_f)$ that contains no constraint belonging to a faulty component."

3.3. Objective reconfiguration

Objective reconfiguration is the third FH level, it is associated with non recoverable faults. In that case, the question is "is there a configuration (H,S) and a property P^* of interest such that $P^*(H,S)$?

Consistent Objective: An objective P^* is consistent if there exists a configuration (H,S) that is valid for it.

Objective reconfiguration: Objective reconfiguration (OR) is the strategy by which a non recoverable fault is handled by changing the current USOM into a new USOM associated with a consistent objective.

Remark 6: (1) Objective reconfiguration must be used when non recoverable faults occur, but it may be of interest even when the fault is recoverable. (2) There is a finite number of objectives the system can be required to achieve. They are generally pre-calculated and define the set of the system operating modes (USOM), for example activating the security shut-down mode reconfigures the production objecttive into a emergency stop one. In some cases objecttives are parameterized, and objective reconfiguration may be limited to the on-line calculation of new parameters (as for path replanification in robotics applications). (3) In a particular fault situation, the existence of a consistent functional objective is a fault tolerance issue, while the existence of a consistent safety objective is indeed a safety issue.

Safe system: The system is safe for a given non recoverable fault H_f, if there exists a consistent reconfiguration to a safety objective (an emergency stop, for example).

4. A FAULT TOLERANT ESTIMATION EXAMPLE

4.1. System description

In order to illustrate the above concepts, consider a system with 4 unknown variables $\{x_1,...x_4\}$ and 5 sensors $\{y_1,...y_5\}$. The state vector x and the output vector y obey state and output equations given by

$$\dot{x} = f(x,u),$$
$$y = g(x).$$

In closed-loop operation, the control is given by a control law

$$u = h(x),$$

which results in

$$\dot{x} = f(x,h(x)), \tag{7}$$
$$y = g(x). \tag{8}$$

Following a classical approach, the estimation algorithm is built from successive derivations of (8) where the state derivatives are replaced by their expression from (7) until the obtained set of constraints allows the computation of the whole state [14]. Assume here that this process provides 7 constraints $\{c_1,...c_7\}$ which link the unknown states and the derivatives of the known sensor outputs, and which, by the system hardware architecture, are associated with 3 components $\{\sigma_1,\sigma_2,\sigma_3\}$ as follows :

$$\begin{aligned}
&\sigma_1 \begin{cases} c_1(x_1,x_2,x_3)=0 \\ c_2(x_1,x_4,\bar{y}_2)=0 \end{cases} \\
&\sigma_2 \begin{cases} c_3(x_4,\bar{y}_5)=0 \\ c_4(x_1,x_2,\bar{y}_1)=0 \\ c_5(x_3,\bar{y}_2,\bar{y}_3)=0 \end{cases} \\
&\sigma_3 \begin{cases} c_6(x_4,\bar{y}_2,\bar{y}_3)=0 \\ c_7(x_4,\bar{y}_4)=0. \end{cases}
\end{aligned} \tag{9}$$

4.2. Objective and valid configurations

Consider a USOM where the objective is to control x_1 using an estimate $\hat{x}_1$. Assume the estimation is needed with a specified accuracy α_0. Structural Analysis is a simple means to identify the different computation means available to perform this estimation [16].

Define Level_1 as the set of those variables that can be directly computed from the sensor signals : there are three different means to estimate x_4 (3 versions)

$$c_3(x_4,\bar{y}_5)=0 \rightarrow \hat{x}_4^1=\gamma_3(\bar{y}_5)$$
$$c_6(x_4,\bar{y}_2,\bar{y}_3)=0 \rightarrow \hat{x}_4^2=\gamma_6(\bar{y}_2,\bar{y}_3)$$
$$c_7(x_4,\bar{y}_4)=0 \rightarrow \hat{x}_4^3=\gamma_7(\bar{y}_4)$$

and one single means to estimate x_3

$$c_5(x_3,\bar{y}_2,\bar{y}_3)=0 \rightarrow \hat{x}_3^1=\gamma_5(\bar{y}_2,\bar{y}_3),$$

where $\gamma_i(.)$ is the result of solving constraint $c_i(.)=0$ with respect to the selected unknown.

Level_2 contains the variables that can be computed using only sensor outputs and Level_1 variables. At this level, x_1 can be estimated by solving c_2 (under 3 versions)

$$c_2(x_1,x_4,\bar{y}_2)=0 \rightarrow \hat{x}_1^{1,2,3}=\gamma_2(\hat{x}_4^{1,2,3},\bar{y}_2)$$

and x_2 can be estimated by solving c_4

$$c_4(x_1,x_2,\bar{y}_1)=0 \rightarrow \hat{x}_2^{1,2,3}=\gamma_4(\hat{x}_1^{1,2,3},\bar{y}_1).$$

Therefore, the 4 variables are observable. However, the shortest computation paths given above are not the only ones, indeed x_3 can also be computed by solving c_1 provided x_1 and x_2 have been estimated before

$$c_1(x_1,x_2,x_3)=0 \rightarrow \hat{x}_3^2=\gamma_1(\hat{x}_1,\hat{x}_2)$$

and x_1 and x_2 can be estimated together by solving the system of equations

$$\begin{cases} c_1(x_1,x_2,\hat{x}_3^1)=0 \\ c_4(x_1,x_2,\bar{y}_1)=0 \end{cases} \rightarrow \begin{pmatrix} \hat{x}_1^4 \\ \hat{x}_2^4 \end{pmatrix}=\gamma_{14}\left(\hat{x}_3^1,\bar{y}_1\right)$$

($\gamma_{ij}(.)$ is the solution of the system $c_i(.)=0$, $c_j(.)=0$). Finally there exist 4 configurations that allow to estimate x_1. They are shown in Table 1.

Table 1. The four possible configurations for the estimation of x_1.

Configuration components constraints	Algorithm
$(H,S)_1$ σ_1,σ_2 c_2,c_3	$\hat{x}_1^1=\gamma_2(\gamma_3(\bar{y}_5),\bar{y}_2)$
$(H,S)_2$ σ_1,σ_3 c_2,c_6	$\hat{x}_1^2=\gamma_2(\gamma_6(\bar{y}_2,\bar{y}_3),\bar{y}_2)$
$(H,S)_3$ σ_1,σ_3 c_2,c_7	$\hat{x}_1^3=\gamma_2(\gamma_7(\bar{y}_4),\bar{y}_2)$
$(H,S)_4$ σ_1,σ_2 c_1,c_4,c_5	$\hat{x}_1^4=\gamma_{14}(\gamma_5(\bar{y}_2,\bar{y}_3),\bar{y}_1)$

Let $\alpha_i, i=1,..4$ be the precision associated with estimation versions $i=1,..4$, and assume that $\alpha_1 \le \alpha_2 \le \alpha_3 \le \alpha_0 < \alpha_4$, it is concluded that only configurations $(H,S)_{1,2,3}$ are valid in the considered USOM.

4.3. Fault handling

Assuming that $(H,S)_1$ is the configuration currently in use, let us illustrate the different Fault Handling concepts.

Fault modes: Any subset of the 7 constraints can be falsified, therefore there are $2^7-1=127$ fault modes. For example, there are 3 fault modes of component σ_1 (c_1 false, c_2 false, both c_1 and c_2 false), 7 fault modes of component σ_2 and 3 fault modes of component σ_3, giving a total of 13 "single component" fault modes and 114 "multiple components" fault modes.

Passive Fault Tolerance is associated with the question "is the current configuration $(H,S)_1$ still valid after the fault occurrence"? this is obviously true as long as the fault mode does not contain constraints c_2,c_3. Therefore, there are $2^5-1=31$ fault modes in which the estimation of x_1 can still be carried on with the desired accuracy. Note that in order to decide in real time whether this is the case or not, it is needed not only to detect the fault but also to isolate the faulty constraints. Note also that if only the faulty components (instead of the faulty constraints) are isolated by the FDI algorithm, PFT is known to be achievable only in 3 fault modes although it is really achievable in 31 fault modes (indeed, $(H,S)_1$ is known to be valid in those cases where component σ_3 alone is faulty).

Assume now that a fault mode that affects constraints c_2 and/or c_3 occurs, and let $\tilde{H}=\{\tilde{c}_2, \tilde{c}_3\}$ be the faulty constraints. Fault Accommodation is associated with the question "is there an estimation software $\tilde{S}$ such that $(\tilde{H},\tilde{S})$ is valid"? it obviously follows that for the fault $\tilde{c}_2,\tilde{c}_3$ to be FA-recoverable it is necessary (but not sufficient) that both constraints are still invertible, leading to the accommodated estimation algorithm $\hat{x}_1^1=\tilde{\gamma}_2(\tilde{\gamma}_3(\bar{y}_5),\bar{y}_2)$. Note that in order to decide whether $\tilde{c}_2,\tilde{c}_3$ are still invertible and to synthesize the accommodated estimation algorithm, these faulty constraints must be not only detected and isolated, but also they must be identified. A necessary and sufficient condition for the fault to be

FA-recoverable is that constraints $\tilde{c}_2, \tilde{c}_3$ are both invertible and moreover, their inversion allows for the accuracy specification to hold (for example they are identified precisely enough). If this condition does not hold, one must turn to System Reconfiguration. The question now becomes : "is there a configuration (H,S) with $H \cap \tilde{H} = \varnothing$ which is valid? it is easily seen that $(H,S)_2$ or $(H,S)_3$ both answer the question provided the fault is only on c_3, otherwise there is no solution. Note that to answer this question, only detection and isolation are needed.

Going further with the example, assume that the FDI algorithm provides only detection and isolation, and that c_2 is faulty, then there is no means to recover from the fault, neither by FA (since no fault model is available) nor by SR (since no healthy configuration is valid). The objective to estimate x_1 with accuracy better than α_0 cannot be achieved, unless performance degradation is allowed up to α_4 : in that case configuration $(H,S)_4$ can be used. Otherwise, the objective must be reconfigured by firing a transition from the current USOM (regulate x_1 using its estimate $\hat{x}_1$), to another one where the estimation is not needed, for example enter an idle or an emergency stop operating mode.

5. FAULT HANDLING PERFORMANCES

Based on the previous concepts, this section analyzes how the fault handling properties of a system can be evaluated.

5.1. Fault tolerance

Remember that fault tolerance is defined with respect to a given property P, an admissible degradation of P, which is specified by the weaker property P^- and a given set of fault modes F: system Σ is fault tolerant with respect to the triple (P,P^-,F) if any fault $f \in F$ can be recovered, i.e., there exists a configuration - namely (H_f,S_n) for PFT, (H_f,S_f) for FA and $(\bar{H}_f,S_f)$ for SR - that allows to achieve the - maybe degraded - objective P.

It immediately follows that the only sound way to evaluate the fault tolerance of a system Σ for a pair (P,P^-) is to measure the set of recoverable faults. Define $\mathcal{F}_{PFT}$ the set of faults that are passively recoverable, $\mathcal{F}_{FA}$ the set of faults that are recoverable by fault accommodation and $\mathcal{F}_{SR}$ the set of faults that are recoverable by system reconfiguration. The set of recoverable faults is $\mathcal{F} = \mathcal{F}_{PFT} \cup \mathcal{F}_{FA} \cup \mathcal{F}_{SR}$ and Σ is all the more fault tolerant for the pair (P,P^-) as this set is "big".

This approach obviously needs a measure to be defined. Defining such a measure is not straightforward for $\mathcal{F}_{PFT}$ and $\mathcal{F}_{FA}$ since a given subset of faulty constraints may be recoverable or not, depending on the value of the parameters in the model H_f, as can be seen from the fault case where constraints c_2 and/or c_3 are faulty in the above example : the accommodated estimation algorithm $\hat{x}_1^1 = \tilde{\gamma}_2(\tilde{\gamma}_3(\bar{y}_5),\bar{y}_2)$ might be accurate enough only for some values of the fault parameters, i.e., only for a subset of all possible faulty constraints $\tilde{c}_2, \tilde{c}_3$.

The definition is much simpler when restricted to the set $\mathcal{F}_{SR}$ since in that case, whatever the value of the fault parameter associated with a faulty constraint, this constraint is eliminated from the hardware configuration (this obviously implies a decrease of the fault tolerance performance). In the sequel, we investigate the evaluation of fault tolerance only when SR strategies are used.

5.1.1 Functional critical faults

At time t, the system constraints can be decomposed into two classes, the healthy ones (index 0) and the faulty ones (index 1)

$$C = C_0(t) \cup C_1(t),$$

where only the constraints in $C_0(t)$ are usable by a software configurations S_f.

Functional critical fault: Given a pair (P,P^-), a functional critical fault is a *minimal subset of constraints* $\bar{C} \subseteq C$ such that no valid configuration for this pair exists, whose hardware constraints would be included in $C \setminus \bar{C}$.

Let $\bar{C}_{cr}(P,P^-) = \{\bar{C}_{cr}^1, \bar{C}_{cr}^2, \ldots\}$ be the set of all functional critical faults for the pair (P,P^-).

Example 1: In the previous example, assume that performance degradation is allowed up to α_4. Then, all the 4 versions of the estimation algorithm given by Table I are admissible. The Boolean expression

$$(P,P^-) = c_2 \cdot c_3 + c_2 \cdot c_6 + c_2 \cdot c_7 + c_1 \cdot c_4 \cdot c_5,$$

where $c_i.c_j$ (resp. $c_i + c_j$) are condensed notations for $\mathcal{H}_0(c_i) \wedge \mathcal{H}_0(c_j)$ (resp. $\mathcal{H}_0(c_i) \vee \mathcal{H}_0(c_j)$) expresses the conditions under which a valid configuration for property (P,P^-) exists. It follows that the set of functional critical faults is

$$\bar{C}_{cr}(P,P^{-}) = \{\{c_1,c_2\},\{c_2,c_4\},\{c_2,c_5\},\\ \{c_1,c_3,c_6,c_7\},\{c_3,c_4,c_6,c_7\},\\ \{c_3,c_5,c_6,c_7\}\}. \tag{10}$$

5.1.2 Evaluating the fault tolerance

Using a SR strategy, property (P,P^{-}) is fault tolerant as long as

$$\forall \bar{C}^{i}_{cr} \in \bar{C}_{cr}(P,P^{-}) : \bar{C}^{i}_{cr} \not\subset C_1(t).$$

Since $\bar{C}_{cr}(P,P^{-})$ is determined off-line, one can easily check on-line, when components fail and are switched-off by the reconfiguration procedure, if the set $C_1(t)$ satisfies this condition. Even better, it is possible to evaluate the "remaining fault tolerance" of (P,P^{-}), at time t, as follows.

Redundancy degrees: The cardinal of each element of $\bar{C}^{k}_{cr} \in \bar{C}_{cr}(P,P^{-})$, $k = 1,2,\ldots$ is the *number of constraints whose violation is sufficient* to falsify (P,P^{-}). An evaluation of the remaining fault tolerance level at time t, is therefore given by the number of components (remember that each component is associated with a set of constraints) whose switching-off would lead to a set $C_1(t)$ such that :

$$\exists \bar{C}^{i}_{cr} \in \bar{C}_{cr}(P,P^{-}) : \bar{C}^{i}_{cr} \subseteq C_1(t).$$

The smallest number of such components is called "strong redundancy degree", the largest one is the "weak redundancy degree" [13,14]. Strong and weak redundancy degrees evaluate the remaining fault tolerance by the number of components (respectively in the "pessimistic" and the "optimistic" case) whose fault falsifies the expected property. However, more elaborate indexes can be used, since faults are events whose probability can be evaluated, provided reliability data are available.

Reliability and mean time to falsification: The fault tolerance of property (P,P^{-}) can be evaluated through its reliability $R(C_0(t),t,T)$, which is the probability for (P,P^{-}) to remain achievable on the time interval $[t,T]$, under initial non-falsified constraints $C_0(t)$ at time t. A complementary index is $MTF(C_0(t),t)$, the mean time to the falsification of property (P,P^{-}), under the initial condition $C_0(t)$ at time t.

Example 2: Remember that the 7 constraints in (9) model a system of 3 interconnected components $\{\sigma_1,\sigma_2,\sigma_3\}$. Assume that system reconfiguration by switching-off faulty components is the only FT strategy that is used. Fig. 1 shows for each configuration of in-service components (identified on the first line of the box), the set of available versions of the estimation algorithm of x_1 (on the second line of the box). Top-down arrows show configuration changes that result of switching-off components after faults, while bottom-up arrows show configuration changes that result from switching-on the components again (e.g., after repair).

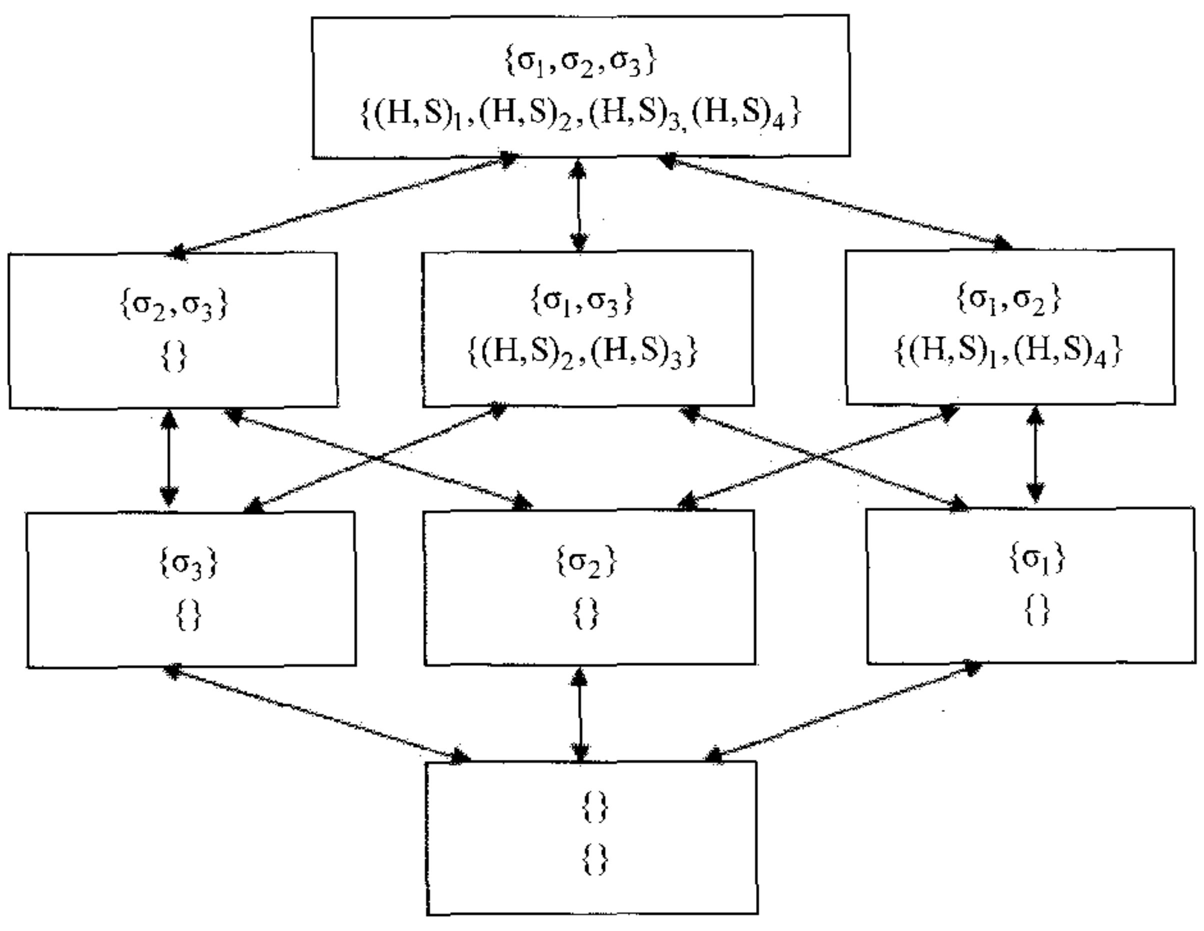


Fig. 1. Possible configurations and available estimation versions.

In this system, the strong redundancy degree is 1, since the single fault of σ_1 is enough to falsify (P,P^{-}); the weak redundancy degree is 2 since there are cases where two components can be faulty before the estimation becomes impossible. On another hand, if all three components are assumed to be healthy at time 0 and their reliabilities are known, the reliability of the estimation function of x_1 is computed by:

$$\begin{aligned} R(\{\sigma_1,\sigma_2,\sigma_3\},0,T) &= \Pi_{[0,T]}(\sigma_1,\sigma_2,\sigma_3) \\ &\quad + \Pi_{[0,T]}(\sigma_1,\sigma_3)\left[1 - \Pi_{[0,T]}(\sigma_2)\right] \\ &\quad + \Pi_{[0,T]}(\sigma_1,\sigma_2)\left[1 - \Pi_{[0,T]}(\sigma_3)\right], \end{aligned}$$

where $\Pi_{[0,T]}(a,b,\ldots)$ is the probability for the components $\{a,b,\ldots\}$, that are healthy at time 0, to remain healthy during the whole time interval $[0,T]$. The mean time to falsification, $MTF(\{\sigma_1,\sigma_2,\sigma_3\},0)$ follows using classical computations [14].

5.2. Safety performances

Safety performances can be analyzed in a similar way to functional performances, by considering only the case where the current objective must be reconfigured to a safety objective.

Safety critical fault: A safety critical fault is a minimal subset of constraints $\bar{C} \subseteq C$ such that there is no valid configuration whose hardware constraints would be included in $C \setminus \bar{C}$ that allows to achieve any safety objective.

Although the existence of possible objective reconfiguration is more difficult to assess than the existence of valid configurations for the current objective, the definition of safety critical faults is merely the same as the definition of functional critical faults, and therefore the evaluation process is completely similar.

5.3. Fault avoidance

Since both fault tolerance and safety can be evaluated by using reliability measures, it obviously follows that an increase in these quantities can be obtained by a decrease in the probabilities for the components to be faulty.

Fault avoidance is the strategy by which the components reliability is increased, either by selecting top-quality components, or by implementing an improved maintenance policy.

6. CONCLUSION

This paper has proposed an overview of the different concepts associated with fault handling in control systems, in an attempt to draw a clear map showing the structure and proposing an organization of this recent field of research. The presentation was done at the ontological level, by distinguishing different Fault Tolerance strategies, setting the frame in which Fault Tolerance, Fault Avoidance, and Fault Handling complement each other, and analyzing how these properties can be ascertained and evaluated for a given design. The ontological level that was presented does not address the design and optimization of fault handling procedures from a real time perspective, whose constraints are also important in practical application. The problems associated with this point of view are : how to minimize fault detection and isolation delays, how to identify quickly and precisely the model of the faulty system (when needed), how to minimize the fault handling delays, how to take into account diagnostic errors, or errors in the application of the recovery procedures themselves, etc. This quite impressive list shows that the field still needs a huge research effort.

REFERENCES

[1] G. Biswas, M. Cordier, J. Lunze, L. Travé-Massuyes, and M. Staroswiecki, "Diagnosis of complex systems: Bridging the methodologies of the FDI and DX communities," *IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics*, vol. 34, no. 5, pp. 2159-2162, May 2004.

[2] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and Fault Tolerant Control*, Springer-Verlag, Berlin-Heidelberg, Germany, 2003.

[3] J. de Kleer, A. Mackworth, and R. Reiter, "Characterizing diagnosis and systems," *Readings in Model Based Diagnosis*. Morgan-Kauffman Pub., San Mateo. pp. 54-65, 1992.

[4] M. I. Brudny, *Popper, un Philosophe Heureux*, Grasset, Paris, 2002.

[5] A. L. Gehin and M. Staroswiecki, "A formal approach to reconfigurability analysis. application to the three tank benchmark," *Proc. of European Control Conference*, pp. 1039-4, 1999.

[6] A. L. Gehin and M. Staroswiecki, "Reconfiguration analysis using generic component models," *IEEE Trans. on Systems, Man, and Cybernetics. Part A: Systems and Humans*, vol. 38, no. 3, pp. 575-583, May 2008.

[7] J. Gertler, *Fault Detection and Diagnosis in Engineering Systems*, Marcel Dekker, Inc., Basel, 1998.

[8] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," Presented at the Padua workshop on Formal Ontology, March 1993, later published in *International Journal of Human-Computer Studies*, vol. 43, no. 4-5, pp. 907-928, November 1995.

[9] R. Isermann, *Fault-Diagnosis Systems*, Springer, Berlin, Germany, 2006.

[10] J. Jiang and Y. M. Zhang, "Accepting performance degradation in fault-tolerant control system design," *IEEE Trans. on Control Systems Technology*, vol. 14, no. 2, pp. 284-292, 2006.

[11] M. Mahmoud, J. Jiang, and Y. M. Zhang, *Active Fault Tolerant Control Systems: Stochastic Analysis and Synthesis*, vol. LNCIS-287. Springer, Berlin, Germany, 2003.

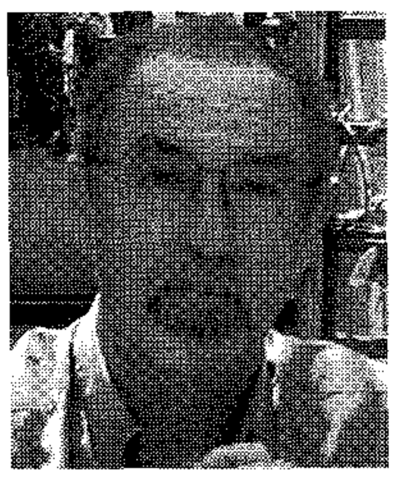
[12] K. R. Popper, *Conjectures and Refutations*. Routledge and Kegan Paul, London, 1963.

[13] M. Staroswiecki, "On fault tolerant estimation in sensor networks," *Proc. of European Control Conference*, Cambridge, UK, 2003.

[14] M. Staroswiecki, G. Hoblos, and A. Aitouche, "Sensor network design for fault tolerant estimation," *International Journal of Adaptive Control and Signal Processing*, vol. 18, pp. 55-72, 2004.

[15] M. Staroswiecki, "Robust fault tolerant linear quadratic control based on admissible model matching," *Proc. of the 45th IEEE Conf. on Decision and Control*, pp. 3506-3511, 2006.

[16] M. Staroswiecki, "Observability and the design of fault tolerant estimation using structural analysis," *Advances in Control Theory and Applications*, Series: Lecture Notes in Control and Information Sciences, vol. 353, pp. 257-278. C. Bonivento, A. Isidori, L. Marconi and C. Rossi, (Eds.), 2007.
[17] Y. M. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *Proc. of IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, pp. 265-276, 2003.
[18] X. Zhang, T. Parisini, and M. M. Polycarpou, "Adaptive fault-tolerant control of nonlinear uncertain systems: An information-based diagnostic approach," *IEEE Trans. on Automatic Control*, vol. 49, no. 8, pp. 1259-1274, 2004.



Marcel Staroswiecki received the Mechanical Engineer degree from Ecole Nationale Superieure des Arts et Metiers, Paris (France) in 1968. His PhD and Habilitation were obtained from University of Lille, respectively in 1972 and 1981. His research has been dedicated to Fault Diagnosis and Fault Tolerance for more than twenty years. He is currently a Professor with Ecole Polytechnique Universitaire de Lille for teaching and with Laboratoire SATIE, at Ecole Normale Superieure de Cachan for research.