AN INTEGRATED DESIGN APPROACH TO MULTILEVEL FAULT TOLERANT CONTROL OF DISTRIBUTED SYSTEMS [1]

Claudio Bonivento [*] **Marta Capiluppi** [*]
Lorenzo Marconi [*] **Andrea Paoli** [*,2]

[*] *Center for Research on Complex Automated Systems (CASY) "Giuseppe Evangelisti", DEIS, Department of Electronic,Computer Science and Systems, University of Bologna, Via Risorgimento 2, 40136 Bologna, Italy*

Abstract: This work deals with the description of a design procedure for hierarchical Fault Tolerant Control (FTC) of large-scale systems. Following a functional perspective, a procedure for the modular design of the diagnostic and reconfiguration algorithms which run at different levels of the hierarchy is presented. Moreover a hierarchical decision logic algorithm is obtained by means of the theoretical machinery of the supervisor theory of discrete event systems. All the material presented here represents a brief summary of the ideas and results obtained in the context of the European Project IFATIS (proposal number IST-2001-32122). *Copyright © 2005 IFAC*

Keywords: Large-scale systems, Fault-tolerant systems, Fault diagnosis, Complex systems, Supervisory control.

1. INTRODUCTION

A (real-time) *distributed (or large scale) system* consists of a set of *nodes* interconnected by a *real-time communication network*. Viewed from an higher level, a node can be replaced by an abstraction of its functional and temporal properties, hiding the irrelevant details of the implementation. Nowadays in complex control applications the distributive aspect has grown in importance. From a functional point of view, there is practically no difference whether a task is implemented using a centralized or decentralized architecture, however a decentralized architecture has to be preferred for the implementation of hard real-time systems. The property which makes distributed systems perfect candidate for investigation in fault tolerance area is *dependability.* This means that, in the design of a distributed system, it is possible to implement well-defined error-containment regions, achieving in this way fault tolerance. An exhaustive description of distributed systems can be found in (Koepetz, 1997), while, concerning the topic of fault-tolerance in distributed systems, a good introduction to the problem can be found in (Lee and Anderson, 1990), (Johnson, 1989), (Jalote, 1994) and (Cristian, 1991).

In distributed control systems every component must provide a certain function in order to make the overall system working satisfactorily. When a fault affects the system, then it causes a failure, i.e. the termination or degradation of the ability of

[1] This research has been supported by EC-Project IFATIS (intelligent fault tolerant control in integrated systems) funded by the European Commission in the IST programme 2001 of the 5th EC framework programme (IST-2001-32122).

[2] Corresponding author Dr. Andrea Paoli, E-mail: `apaoli@deis.unibo.it`.

an item to perform its required function. We refer as failure mode to the effect by which a failure is observed on the failed system. This means that a failure mode represents a loss of a functionality. In distributed systems a single component failure, propagating into the system structure, may lead to a catastrophic system failure. For this reason fault tolerance (i.e., as explained in (Blanke *et al.*, 2003), the property of performing a required function with predefined performances even in faulty situations) is a key issue in distributed systems. Fortunately the distributive aspect of these systems helps in achieving the fault tolerance task: if there exists a one-to-one mapping between functions and nodes, the cause for a malfunction can be immediately diagnosed and the faulty node isolated. Weak elements in this framework are control loops: automated systems are vulnerable to faults such as defects in sensors, in actuators and in controllers, which can cause undesired reactions and consequences as damage to technical parts of the plant, to personnel or to the environment. Then the design of a Fault Tolerant Control (FTC) architecture is of crucial importance and solutions aiming at adapting the control strategy to the presence of the faults are needed in order to achieve prescribed performances also for the faulty system.

This work attempts to present a unified framework for fault tolerant control of distributed systems summarizing some of the ideas and results obtained in the context of the European Project IFATIS. Following a functional approach, a modular/hierarchical fault tolerant control architecture is presented and some design guidelines which make use of standard failure and functional analysis tools are presented. An example of application to a physical system of the general concepts illustrated in this work is illustrated in (Bonivento *et al.*, 2005).

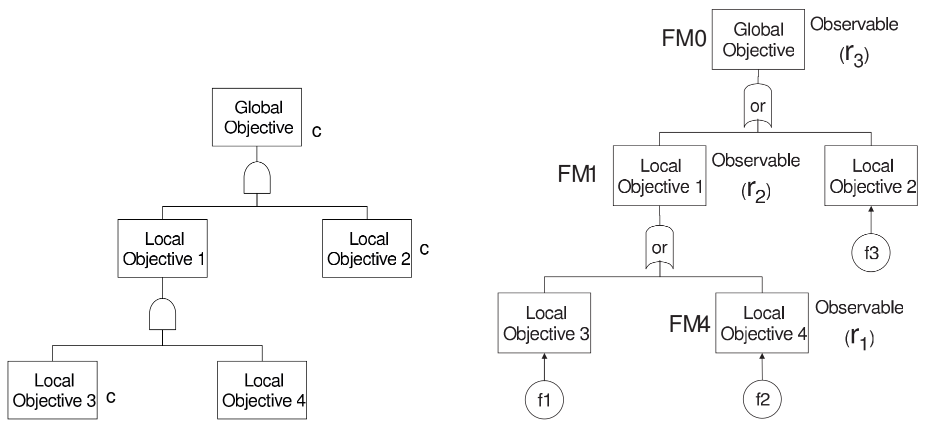
2. GENERAL FRAMEWORK

2.1 IFATIS Architecture

According to the description given in (Bahir *et al.*, 2003) and (Maier and Colnaric, 2002), the overall distributed system can be divided in partial processes, a set of physical/logical controlled systems whose aim is to achieve a certain functionality instrumental for the operation of the whole system. Within each partial process a physical level and a control level (denoted as fault tolerant module) exist. This control level aims to control the partial process and to manage the set of possible working modes associated to it in order to achieve the main functionality (completely or in a degraded form if a fault occurs). The implementation of each partial process needs allocation of resources, i.e. plant components and controller modules. This allocation is dynamic, dependent on mode and reconfiguration decisions. The hierarchical architecture for fault tolerant control of such a system consists of three modular levels: the plant level, the control level, the supervision level. At control level *Fault Tolerant modules* are considered. Each module has a specific function to achieve. Each function can be performed in different working modes (for example nominal working mode or reconfigured/degraded working modes). Note that *Fault Tolerant module* stands for a possible hierarchical structure of modules. Above this level there is a supervision level, called Global/Group Resource and Reconfiguration Manager (GRRM) level, whose aim is to monitor performances of the system and to manage physical resources.

2.2 Supervisory control of Discrete Event Systems

Talking about supervisory control of a discrete event system (DES) means dealing with the problem of a given DES (considered in the un-timed level of abstraction), whose behavior must be modified by feedback control in order to achieve a set of given specifications. In other words the problem consists in designing a discrete event supervisor S which is able to restrict the behavior of the uncontrolled system G to a desired subset specified by specifications. The interaction paradigm between S and G is very simple: the supervisor observes some (possibly all) of the events that G executes. Then it tells G which events are allowed next. More precisely S has the capability of disabling some feasible events of G. In this sense S exerts a dynamic feedback control on G. In general there will be some events that cannot be disabled by the supervisor, i.e. they are uncontrollable. A supervisor is said to be admissible if it never tries to disable an uncontrollable event. As mentioned above, the supervisor is introduced in order to restrict the unsatisfactory behavior of the uncontrolled DES to a given specification. In order to synthesize supervisors the starting point is to define an automaton H which expresses the specification for the controlled DES. There are several kind of specifications arising in applications. Besides others, very common are illegal states specifications, event alternance specifications and illegal string specifications. Given an uncontrolled DES G and a specification automaton H, the specification is said to be controllable with respect to G if H contains all the part of G that cannot be prevented because uncontrollable. In the same way H is said to be observable with respect to G if H requires the same control action for two strings that cannot be separated because of unobservable events.There exists a theorem (known in literature as **COT**, i.e. controllability and observability the-



(a) Example of functionality tree with reconfigurable functionalities (label c).

(b) Example of fault tree for diagnosis with diagnosable failure modes (label r_i).

Fig. 1. Example of the use of fault tree analysis to design a distributed control system.

orem) which states that if and only if H is observable and controllable with respect to G, then there exists a dynamic feedback supervisor restricting the behavior of G to H. If this is the case, the next step is to build a convenient representation of S. It is easy to prove that if the COT holds then an automaton representation of S is given by H itself and the feedback interconnection between G and S is given by the standard parallel composition operation $H \parallel G$. More details on this theory can be found in (Cassandras and Lafortune, 1999).

3. CRITERIA AND DESIGN TOOLS

As stated in previous sections a functional criterion to design the distributed FTC system has been used. Due to its distributed aspect, partial processes composing the system may be complex systems as well, and the main functionality may be achieved by the fulfillment of several nested sub-functionalities. For this reason it is possible to highlight the functionality map linked to each partial process using a *Functionality Tree* (see (Andrews and Moss, 2002)). This is a graphical tool by which the global functionality of the system, the root of the tree, is expressed in terms of sub-functionalities (i.e. intermediate nodes of the tree) instrumental for the achievement of the main functionality. Strictly related to the functionality tree is the *Fault Tree* which shows the map of losses of functionalities as a consequence of faulty conditions (see (Andrews and Moss, 2002), (Blanke *et al.*, 2003)). The fault tree can be interpreted as a complementary version of the functionality tree. Losses of functionalities at each level of the tree are seen as caused by losses of functionalities sitting at lower levels until the main elementary cause, given by the physical fault, is reached.

The fault and functionality tree provide a useful tool for the design of both diagnostic and reconfiguration algorithms. The problem of diagnosis is strictly connected to the ability of detecting a possible loss of functionality due to a fault. In this respect it makes sense to define failure modes as the loss of functionalities (nodes of the fault tree) which are detectable by static as well as dynamic elaboration of the available measures. In general it is possible to associate to each failure mode in the fault tree one or more residual signals (see Fig. 1(b)). In this sense the functionalities hierarchy described in the functionality tree is reflected in the hierarchy of the diagnostic system aimed to detect the losses of these functionalities. In this respect the whole residual matrix can be constructed through a bottom-up procedure by inspection of the fault tree.

As far as the reconfiguration problem is concerned, the graphical description given by the functionality tree can be used to identify a hierarchy also in the reconfiguration algorithm (see Fig. 1(a)). To this end we define *Reconfigurable Functionalities* as those functionalities on which the designer has some degree of controllability. It is possible to identify a set of control functions associated to each reconfigurable functionality, namely a set of reconfiguration actions instrumental to implement the reconfiguration of the specific functionality. A possible tool to identify reconfigurable functionalities is structural analysis (see (Staroswiecki *et al.*, 1999)). Linked to each reconfigurable functionality, there exists a local supervisor which has the role of supervising the reconfiguration actions to be launched in the case that a specific functionality has been selected for reconfiguration. In particular, linked to each reconfigurable functionality, there is a set of local Working Modes (WM) and a set of local Control Functions implementing a number of control reconfigurations. Aim of the local supervisor is to choose the reconfigurable functionality which is the closest to the estimated failure mode. As a matter of fact the rationale behind this criterion

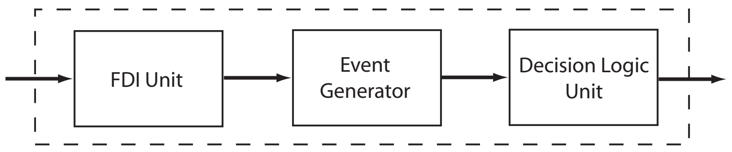


Fig. 2. Supervisor structure.

is to reconfigure the system as close as possible to the original cause of the fault condition, in order to shrink the region in which the deviation from nominal conditions is compacted. According to this description, each local reconfiguration supervisor can be thought as depicted in Fig. 2. The goal of the FDI unit is to provide, on the basis of the different local diagnostic information (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 unit whose aim is to raise requests of working mode changes. All the requests of WM changes issued by local supervisor units must be then managed by a decision logic unit at system-level which validates or not the requests according to a global vision of the associated part of process checking the consistency of the different requests raised by the different modules. The theoretical key tool used in the design of this last phase, as described in the next section, is the supervisor theory for Discrete Event Systems (DES).

4. DESIGN OF THE SUPERVISORY SYSTEM

4.1 Low level supervision

The design procedure for the decision logic unit of each local supervisor (intermediate level supervisory system) can be described by the following steps. The step number 0, detailed in the following, is indeed not needed for the supervisor design but only for verification purposes (see (Cassandras and Lafortune, 1999)).

Step 0: Starting from the set of working modes associated to a local reconfigurable functionality, it is possible to build the DES modeling the system, with respect to the considered fault. The initial state (let call it WM0) models the nominal condition for the system. A fault event f, observable but not controllable, moves the system into a faulty state (F). At this point it is possible to force one of the reconfiguration actions (WM) enlightened in the tree. Events named wm_i are controllable and observable events used by supervisors to force a reconfigured working mode (state WM_i). Repeating the procedure for all possible faults affecting the system a DES modeling its behavior is obtained. In general this automaton (named G) will be composed by a nominal state, a set of faulty states and a set of reconfigured states (see Fig. 3 (a)).

Step 1: Consider now the j-th reconfigurable functionality at level i in the functionality tree. According to the specifications behind the reconfiguration of this functionality, it is possible to design a DES composed by a number of states WM_i whose activation is governed by events wm_i. These ones describe the set of reconfiguration actions which must be taken in order to reconfigure the specific functionality and, possibly, functionalities at lower levels. Events wm_i can be, in general, both controllable and not-controllable by the supervisor. Not-controllable events are, for instance, due to implicit reconfigurations which do not need an explicit activation (passive/implicit fault tolerant controllers). This step is illustrated by Fig. 3 (b).

Step 2: The DES resulting from Step 1 can then be systematically modified to design a DES describing the desired controlled behavior if the j-th functionality is reconfigured. In particular the DES designed in Step 1 is completed with an activating prefix given by concatenation of failure mode events (f) and activating events (a_k). The automaton modeling the i-th level specification will be denoted as H_i (see Fig. 3 (c)). Activating event a_i can be used by event generator to set the i-th level reconfiguration.

Step 3: The controlled behavior of the system at level i ($i = 1 \dots n$), denoted with K_i, can be computed as $K_i = H_1 \parallel \dots \parallel H_i \parallel G$. Note that it is not required that H_i by itself is controllable with respect to G, but $H_1 \parallel \dots \parallel H_n$ must result controllable and observable with respect to G. If this is the case the controllability and observability theorem holds (i.e. $K_n = H_1 \parallel \dots \parallel H_n$) and hence there exists a unique modular controller given by the set of n supervisors $H_1, H_2, \dots, H_n$ (see Fig. 3 (d)).

4.2 High level supervision

Last phase of the design procedure is the design of the high level supervisors (Group/Global RRM). In case the reconfiguration of a certain partial process has impact in the resource allocation and/or the change of working mode in a certain module has to be joined to a change of working mode in a different module, then

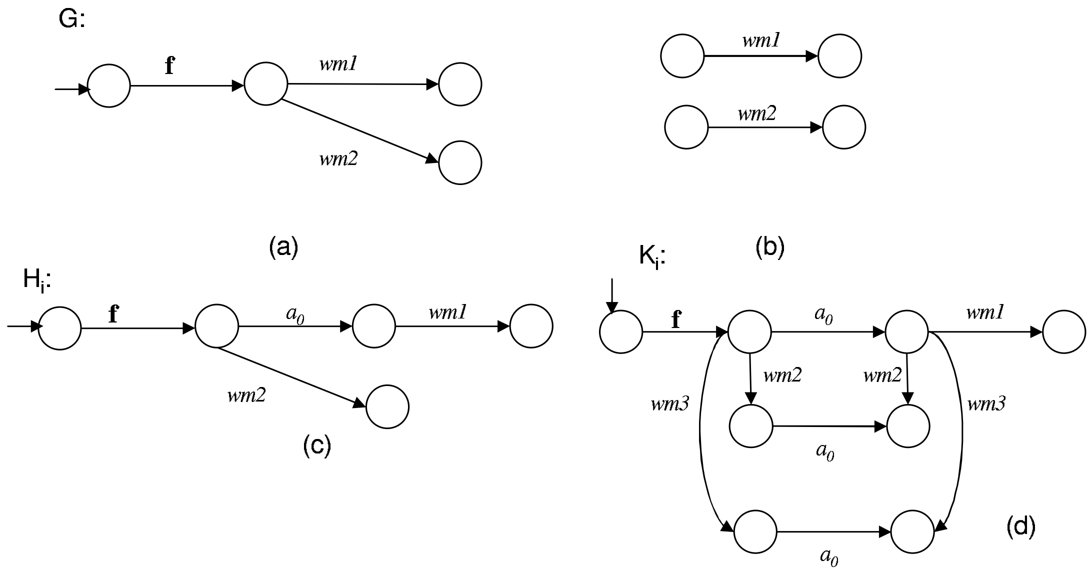


Fig. 3. Design procedure of the low level supervision system.

the responsibility of managing the new working mode switching is demanded to the Group/Global RRM. It is composed just by a Decision Logic Unit which processes all the events generated by the different Local RRM and manages the working mode changes involving reallocations in resources linked to different groups. Moreover the state of the Group and Global RRM can change due to external commands issued by external operators. The starting point is to identify the specifications which are behind the design of the decision logic. These are precisely presented in the following:

Group Selection: description on how the different partial processes and resource units can be grouped (application dependent).

Modules/Resources map: this represents an offline planning on how the different working modes associated to a specific module can be allocated in the available resources.

Partial processes as DES: the outcome of the design phase regarding the Local reconfiguration manager is a DES describing the desired FT behavior of a particular partial process. This is a set of states associated to different failures and reconfigured situations and a set of events describing transitions between states.

Resource Units as DES: this task amounts in describing local resources as automata by specifying states and events inducing transitions between states as follow:

- States: in the simplest case the states describing the status of resources reduce to three: idle (namely the resource is capable to run additional functionalities), busy (no other functionalities can be located on that resources) and faulty (a resource monitor has detected a local fault, for instance of a computer). The busy state can in general be split in several sub-states expressing different cases in which the resource can be busy.
- Events: change of working modes inducing transitions between idle and busy states (these are, for the supervisor, controllable events which, on the basis of the model of the specific resource and the allocation policy followed in the past, commute between the two states); occurred faults, i.e. not-controllable but observable events arising whenever the local resource monitor detects a fault in the supervised resource.

Reconfiguration specifications: this task involves the specifications regarding interlaced reconfigurations between modules. As also remarked above, one of the goal of the Global (Group) RRM is to check the consistency of the functionalities achieved by the different modules and possibly to inhibit the requests of working mode change raised by the Local RRM in case of conflict. The specification at this level regards the identification of possible conflicting Working Modes and consequent remedial actions.

Working modes Urgency: this information is needed whenever a reconfiguration must be actuated in presence of limited resources. It expresses which functionality is the most urgent in order to meet global specifications. The urgency information depends, in general, on the actual working mode.

Starting from this information, it is possible to automatically design the decision logic unit of the group/global supervisor. The composition of the discrete models of the functional units and physical resources, yields an automaton which captures the whole information about the feasible working modes, according to the actual working mode and to the resources availability. From this automaton the decision logic can be designed following the supervision theory on the basis of performance specifications and reconfiguration requirements.

The algorithm to design the Group Global RRM can be described by the following steps:

(1) Replica of the DES associated to a module according to the Modules/Resources map: whenever a particular partial process can be allocated onto different resources, it is necessary to model its image on the particular resource. Since a supervised partial process is described by a DES (K_n), this step can be carried out simply by building a replica of K_n to model its image on the j-th resource (let call this new DES K_n/j).
(2) Since a partial function can be allocated just on a subset of the possible physical resources, it is fundamental to model the possibility that the particular process is in stand by on the considered resource. For this reason the automaton K_n/j is enriched with a new state SB_j and a controllable event called sb_j. If state SB_j is active in K_n/j then n-th module is not running on the j-th resource. Event sb_j is used by the high level supervisor to move process from a resource to another.
(3) The group selection information and the Modules/Resources map together tell which partial processes and which resources compose a group. Hence it is possible to build the i-th group DES model (named GR_i) as a composition of all the possible images of partial processes and all the resources.
(4) Using the information coming from the Modules/Resources map, the Reconfiguration specifications and the working modes urgency it is possible to design a specification controllable with respect to GR_i and, in this way, the i-th Group RRM supervisor (GRSi). In other words the i-th group specifications tell the system how to manage hardware reconfigurations to allocate in an optimal way processes over resources, how to manage hardware faults and how to deal with functionalities conflicts. It is easy to understand that this step is strongly application-dependent.
(5) Finally the controlled behavior of the i-th group can be found as $L_i = GRS_i \parallel GR_i$.

The global behavior of the system is therefore obtained composing all the controlled group behaviors L_i ($L =\parallel_{(i=1\ldots r)} L_i$). The DES obtained in this way represents the global uncontrolled behavior. The Global RRM supervisor behavior is defined when designing a controllable specification with respect to L .

5. CONCLUSIONS

In this work a design procedure for hierarchical Fault Tolerant Control of large-scale systems has been presented. It has been shown how, under a functional perspective, it is possible to identify a procedure for the modular design of the diagnostic and reconfiguration algorithms running at different levels of the hierarchy. Moreover it has been shown how to use the theoretical machinery of the supervisor theory of discrete event systems to design a hierarchical decision logic algorithm.

REFERENCES

Andrews, J.D. and T.R. Moss (2002). *Reliability and Risk Assessment.* Professional Engineering Publishing.

Bahir, L. El, R. Gros, M. Kinnaert, C. Parloir and J.Yamé (2003). Final report on WP2. IFATIS deliverable D2-5.

Blanke, M., M. Kinnaert, M. Staroswiecki and J. Lunze (2003). *Diagnosis and fault-tolerant control.* Springer-Verlag.

Bonivento, C., M. Capiluppi, L. Marconi and A. Paoli (2005). Distributed fault tolerant control of a two-tanks system. Technical report. CASY-DEIS, University of Bologna. Contact person Marta Capiluppi.

Cassandras, C.G. and S. Lafortune (1999). *Introduction to discrete event systems.* Kluwer Academic Publisher.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Comm. ACM* **34**(2), 57–78.

Jalote, P. (1994). *Fault tolerance in distributed systems.* Prenctice Hall. Englewood Cliffs, N.J.

Johnson, B. (1989). *Design and analysis of fault-tolerant digital systems.* Addison Wesley. Reaing, Mass, USA.

Koepetz, H. (1997). *Real-time systems: design principles for distributed embedded applications.* Real-time systems. Kluwer academic publishers. London.

Lee, P.A. and T. Anderson (1990). *Fault tolerance: principles and practice.* Springer Verlag. Vienna, Austria.

Maier, U. and M. Colnaric (2002). Some basic ideas for intelligent control systems design. *Proceedings of the XV IFAC World Congress, Barcelona, Spain.*

Staroswiecki, M., S. Attouche and M.L. Assas (1999). A graphic approach for reconfigurability analysis. *10th Int. Workshop on principle of diagnosis, Loch Awe.*