Reconfiguration analysis using generic component models

Anne-Lise Gehin, Marcel Staroswiecki

Reconfiguration Analysis using Generic Component Models

Anne-Lise Gehin, Marcel Staroswiecki

***Abstract*—This paper presents a formal approach to analyze system reconfigurability, based on a Generic Component Model (GCM), which describes the system from the services provided by its components, and their organization into Operating Modes, in order to achieve specific objectives.**

Following a bottom-up approach, services provided by elementary components are used as resources for services at a higher level. Several versions exist when the same service can be rendered by using distinct sets of resources. Reconfiguration results from the existence of multiple versions, since a faulty resource does not imply loosing the services that use it. A level regulation example shows the effectiveness of the proposed model and tools.

***Index Terms*—Reconfigurability Analysis, Generic Component Model, Fault Tolerant Control, Functional Decomposition.**

I. Introduction

THE increasing demand for safety and reliability calls for the integration of Fault Detection and Isolation (FDI) and Fault Tolerant Control (FTC) issues, at the very early stages of control systems design. A lot of effort has been directed to the design of efficient FDI algorithms [1], [2]. Typically, FDI involves checking the consistency of observations from the real time system operation with prior available, model-based [3], or data-based [4] knowledge. FTC issues have been considered more recently [5]. Typically, the problem is to control to the extent possible the operation of the system in the presence of fault(s). What is meant by control, in this situation, is defined in [6] under three major headings:

1) continue the system operation without (unbearable) loss of performance,
2) continue the system operation with reduced specifications,
3) abandon the mission while avoiding disaster.

A more precise statement of the FTC problem is given in [7], [8], [9]. FDI algorithms, being part of the information system of the supervised process, need information redundancy, whereas FTC being part of the decision and actuation system, is based on decision and actuation redundancy, i.e., on the redundancy of the services which are provided to the users by the process components. The existence of such a redundancy characterizes the reconfigurability property of a system, that is, the potential it has (or not) to continue to carry out its objectives, when some failure occurs [7].

A.-L. Gehin is with the LAGIS UMR CNRS 8146, Université des Sciences et Technologies de Lille, 59655 Villeneuve d'Ascq cedex, France (e-mail: anne-lise.gehin@polytech-lille.fr).

M. Staroswiecki is with the SATIE UMR CNRS 8029, Ecole Normale Supérieure de Cachan, 94235 Cachan cedex, France (e-mail: marcel.staroswiecki@univ-lille1.fr).

Different approaches have been proposed to the design of FTC algorithms under actuator faults [10], [11], [12], or sensor faults [13], [14]. Most of them use the quantitative system behavior model, for example, state and output equations in the time domain. Integrating such approaches in complex systems that involve human operators in the control loop, needs the development of a decision support system to analyze faulty situations and inform the operators about the different possibilities the system still has (or not) to achieve its objectives, in a qualitative way.

In this paper, we propose to analyze system reconfigurability using a generic component model (GCM), first introduced for studying the interoperability of intelligent instruments [15]. This model is based on the notions of service and their organization into User Selected Operating Modes (USOMs). Modelling the component services requires a functional approach, which has been used in the design of control or diagnosis systems [16], [17], [18]. The proposed model includes more general features (e.g. service versions, enabling conditions, operating modes going further than the classical distinction between normal and faulty, etc.), for building an on-line decision support system to analyze the fault tolerance possibilities, and assist the operators in case of failures. It can be applied at any hierarchical level of a system, since it allows for the modelling of high-level components by aggregating low-level models [19].

The paper is organized as follows. The generic component model is presented in Section 2 and used in Section 3 to model the nominal behavior of a system. Section 4 shows its ability to analyze the system reconfiguration possibilities. Section 5 illustrates the proposed approach on a level regulation process. Section 6 presents concluding remarks.

II. The generic component model

The Generic Components Model (GCM) describes components from the point of view of the user, who receives services and can use them in different operating modes. Users may be human operators or other system components, and requests for services can be addressed through direct or remote connection. Interconnections are taken into account by aggregating lower level components into higher level ones. The formal description of GCMs allows for component manipulation in a systematic way at any hierarchical level.

This section introduces the GCM, first using a verbal description, then providing a formalized model.

A. Services

From the user viewpoint, a system component provides one or more services. For example, a level sensor provides

a signal which depends on the level of liquid in a tank. The signal may be validated or not, it may be filtered or not, the sensor might memorize the minimum (the maximum) value in a given time window, or provide an alarm if the signal exceeds a given threshold, and so on. All these are examples of services provided in the normal operating mode. More services could be provided in the installation, initialization, degraded operation, maintenance modes, thus giving the sensor the status of *intelligent* sensor [20].

A service is described by the variables it consumes *(cons)*, the variables it produces *(prod)*, and a procedure *(proc)* which transforms the former into the latter. Services are derived from the components behavior, that is governed by physical laws and (possibly) embedded software. For example, a tank consumes input and puts out mass flows, and produces a stored mass, by the procedure $\dot{m}(t) = q_{in}(t) - q_{out}(t)$, (where m is the stored mass, q_{in} is the flow in the input pipe, q_{out} is the flow in the output pipe), which follows from the conservation of mass; the *regulation* service of a controller consumes signals from sensors and produces signals to actuators according to a given algorithm.

Services may be provided *unconditionally* or on specific *requests*. The *storage* service of a tank is systematically provided (no special request is necessary), at all times and whatever the values of the inputs and outputs; a sensor provides its *measurement* service on a *read* request from a processor. Services may be *enabled* or not, so as to control the conditions under which the requests are accepted: the *read* request should not be enabled when the sensor is known to be faulty or under maintenance. Both the request *(rqst)* and the enabling condition *(ena)* are necessary for a conditional service to be delivered, but the request for a service is issued by the user, while the enabling condition is processed by the control system.

The resources *(res)* of a service are defined as the set of hardware and software elements required to the service realization. Examples of resources are the non leaking tank for a *storage* service, the transducer, filter, analog to digital converter, etc. for the *measurement* service of a sensor.

B. Versions

Services describe what the user expects to obtain from a component under normal operation. However, there are two reasons by which a given service *s* will fail to deliver the appropriate value of the variables it produces:

- Internal faults affect some resources *res* needed by the service. As a result, the actual values of the produced variables *prod* are not those specified by *proc*. A leak in a tank is an example of an internal fault. The procedure $\dot{m}(t) = q_{in}(t) - q_{out}(t)$ does not correctly describe the behavior of a leaking tank since the flow associated with the leak is not taken into account.

- External faults affect the inputs *cons* of the service. A level regulation service is subject to an external fault when the level value it consumes is false, due to a fault in the level sensor, or if its time stamp is outdated, due to a fault in the communication system.

Fault tolerant components integrate multiple instances of a same service, listed as a set of versions. Each version s^j of a service s is a 6-tuple defined as $s^j = \langle cons^j, prod, proc^j, rqst, ena^j, res^j \rangle$.

Note that all versions of the same service share the same request, and produce the same outputs (so they can be interchanged), but they cannot be simultaneously enabled, and at least one among the inputs, procedures and resources is different from one version to another one.

The different versions of a same service may obviously differ by their accuracy, running time, energy consumption, therefore a preference relation can be defined by the designer, the details of which are not developed here. As a result, the set of versions of a service is ordered, the link with enabling conditions being straightforward: at time *t* when service *s* is requested, the version which is enabled is the most preferred one such that all the resources it needs are known to be non-faulty.

C. User selected operating modes

Control systems are expected to achieve different objectives at different times. For example, a level regulation objective makes sense only if the tank has been filled, and a set-point value has been provided to the controller. Therefore, the *level regulation* service should not be enabled when the tank is empty (during some *no-operation* mode), or when it is emptying (in an *end of production* mode), or when it is filling up using the maximum input pump flow (during a *preparation* mode). The *regulation* mode should be preceded by a *preparation* mode, in which not only the tank is filled up but also the *enter set-point* service is enabled. Transition from the *preparation* to the *regulation* mode should be possible only if the tank has really been filled up and the set point has really been entered.

Similarly to software applications that are decomposed into consistent menus, the set of services of a given component is structured into several operating modes, which are associated with specific objectives. These modes are called USOMs *(User Selected Operating Modes)*.

USOMs are defined by the design engineer, taking into account functional and safety specifications. A USOM must obviously contain all the services that are required to achieve its objectives, and it must not contain services that are not allowed to run simultaneously for safety reasons.

More or less standard approaches and methodological tools are available for the analysis of the operating modes of production systems [21].

D. Definition of the generic model

The generic model of a component is now defined.

Definition 1 (Generic model): The generic model of a component is defined by:

1) a deterministic automaton $A(M, \tau, m^0)$ where:

- $M = \{m_i, i \in I_m\}$ is a set of User Selected Operating Modes *(USOMs)*, each of which is represented by a vertex of the automaton,

- $\tau = \{\tau_{ij}, \quad i,j \in I_m\}$ is a set of transitions, each of which is defined by $\tau_{ij} = \{m_i, m_j, c_{ij}\}$ where m_i is the origin USOM, m_j is the destination USOM, and c_{ij} is a firing condition,
- $m^0 \in M$ is the initial USOM, that is the mode where the system stays when it is switched on.

2) a set of services $S = \{s_l, l \in I_s\}$, each of them being a set of pre-ordered versions, $s_l = \{s_l^j, j \in J(s_l)\}$. A version of a service is the 6-tuple $s_l^j =< cons_l^j, prod_l, proc_l^j, rqst_l, ena_l^j, res_l^j >$.

3) USOM and services are linked in the following way:

- each USOM is associated with a subset of services $S_i,\ i \in I_m$, with $\cup_{i \in I_m} S_i = S$
- each USOM is associated with one or several objectives to be achieved.

III. Building systems from components

Systems are built from the interconnection of different components. Indeed, the services delivered by some components consume variables which are produced by services of other components. For example, the value produced by the *measurement* service of a level sensor is consumed by the *regulation* service of a controller, which in turn produces variables which are consumed by the *power modulation* service of the actuator.

In the GCM, interconnections are taken into account by considering higher level components which aggregate lower level ones. Sensors, actuators, process components are at the (lowest) field-level. Hierarchical system architectures are obtained by creating the GCM of aggregated components, at different levels.

A. The system pyramidal structure

The interest of creating high level components is to reason on high level services, ending at the top level of the overall system and its control objectives. Because systems can have components that participate in more than one functionality simultaneously, we use a pyramidal structure (see Fig. 1) rather than a hierarchical one, where a low level component would belong to only one high level component. The number of levels is decided by the designer so as to obtain the view of the system which suits him best. Let $l = 1$ be the lowest decomposition level (field components) and $l = n$ be the highest decomposition level (the system). In a pyramidal structure, the following properties hold $(l \geq 2)$:

- each component of level $l-1$ belongs to at least one component of level l,
- any component of level l includes at least one component of level $l-1$.

Note that such decomposition is not unique, and the result will generally depend on the designer who performs the modeling task. In industrial processes, high level components show themselves, very often, in a way that is evident to operators because they have functional meaning (the steam generator, the catalytic cracking unit, etc.). For example, reasoning about services is easier if the components which are implied in a same regulation loop are grouped together.

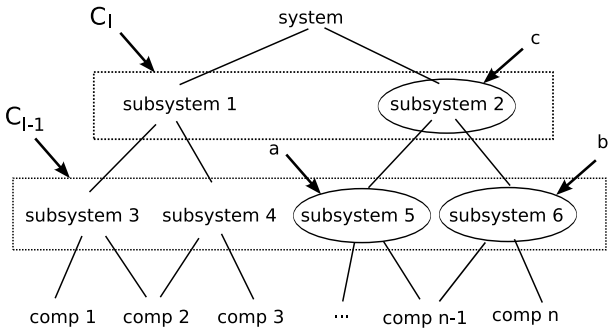

Fig. 1. Pyramidal decomposition of a system.

B. Aggregation of Operation Modes

Let C_l be the set of components modelled at level l $(l = 1, \ldots n)$, let $a, b \in C_{l-1}$ $(l \geq 2)$ and let $c \in C_l$ be a component that aggregates a and b. Let $A(M(a), \tau(a), m^0(a))$ and $A(M(b), \tau(b), m^0(b))$ be the deterministic automata associated with components a and b. The automaton $A(M(c), \tau(c), m^0(c))$ associated with component c is obviously included in the parallel composition of the two automata $A(M(a), \tau(a), m^0(a))$ and $A(M(b), \tau(b), m^0(b))$:

- $M(c) \subseteq M(a) \times M(b)$
- $\tau(c) \subseteq \tau(a) \cup \tau(b)$
- $m^0(c) = \big(m^0(a), m^0(b)\big)$

Indeed, let $\gamma = (\alpha, \beta),\ \gamma \in M(c),\ \alpha \in M(a),\ \beta \in M(b)$. This means that the mode γ of component c is defined as component a being in mode α and component b being in mode β. The parallel composition exhibits all possible modes. However, some of them cannot be given any functional interpretation. For example, the association *(actuator_on, sensor_off, loop_regulation_on)*, is non-significant and can be eliminated. This is done by hand, according to the system specifications, for each aggregated component. The fact that all possible modes are automatically obtained from the parallel composition ensures that no significant mode is forgotten in the aggregation process, while non-significant mode elimination at each step refrains the combinatorial explosion. The process can be assisted by using some rules as in [22].

C. Aggregation of services

Let $S(a)$ and $S(b)$ be the services offered by two components a and b, and let component c be their aggregation. Let $\gamma = (\alpha, \beta)$ be a consistent mode, then any combination of the services $S_\alpha(a)$ (associated with mode α of component a) and $S_\beta(b)$ (associated with mode β of component b) can be used to design services provided by component c. In other words, any "program" using "instructions" from $S_\alpha(a)$ and $S_\beta(b)$ can be a service available in mode γ.

More generally, let Γ be a set of components at level $l-1$ and let c be the component obtained at level l by aggregating all components $k \in \Gamma$. Let

$$cons(c) \quad = \quad \cup_{k \in \Gamma} \cup_{s \in S(k)} cons(s) \tag{1}$$

$$prod(c) \quad = \quad \cup_{k \in \Gamma} \cup_{s \in S(k)} prod(s) \tag{2}$$

(note that $cons(c) \cap prod(c)$ may be non-empty, since some components in Γ may consume variables produced by some

other ones). Let $y_\sigma \subset prod(c)$ and $x_\sigma \subset cons(c)$. Creating a relation between x_σ and y_σ can be done (if needed) by designing a procedure σ such that $cons(\sigma) = x_\sigma$, $prod(\sigma) = y_\sigma$ and $proc(\sigma)$ is a program that use the services $S(k)$, $k \in \Gamma$. Since several subsets of Γ and several procedures over $S(k)\,,\ k \in \Gamma$ could establish the same relation between the variables of interest, several versions might exist. The set of all the versions can be found in a rather automated way, for simple kinds of programs composed of sequences and parallel executions [23]. Once the services of an aggregated component have been designed, ordering their versions follows from cost / quality of service considerations, that are not developed here.

IV. Analysis of fault tolerance using the generic model

Fault tolerance is the possibility of still achieving a given (set of) objective(s) in the presence of a given (set of) fault(s). Therefore, its analysis rests on three points:

1) are there services that allow one to achieve the objectives of the current USOM?

2) how are these services to be managed when faults occur?

3) how are the USOM to be managed when the objectives of the current USOM can no longer be achieved due to the fault(s)?

A. Management of service versions

Remind that a service s is a set of pre-ordered versions: $s = \left\{s^j, j \in J(s)\right\}$. Each version can be used for the same purpose, but the pre-ordering expresses a preference between them. A version s^j of the service s should obviously be disabled when s does not belong to the current USOM. It should also be disabled when some resources in res^j are detected faulty. A disabled version is called *unavailable*. A service is unavailable when it has no available version.

Note that switching from one version which becomes unavailable to another one which is still available creates real time issues (that are not addressed here). Indeed, during the time delay between the occurrence of the fault and the switching of the new version, the faulty system is under nominal control, and that may result in loss of efficiency, (temporary) loss of functionality or - even worse - complete loss of control, if reconfiguration occurs too late. Minimizing the impact of such transients is a major challenge in the design of active Fault Tolerance, that has received little attention in the literature [24], [25], [26]. Although the proposed approach does not specifically address real time issues, it may be noted that ranking the versions of a service according to the criticality of the switching (switching delay, control bumps, etc.) is a possible strategy for reconfiguration.

It is also worth noticing that if a version is not currently running, it will not be allowed to start, even if requested, when disabled. However, if it is currently running, disabling a version does not always stop the service delivery. For example, an on / off valve normally delivers two services, *V_open* and *V_close* (under only one version). If the valve gets blocked closed, both services are disabled i.e. they become unavailable, but *V_open* cannot be run while *V_close* cannot be stopped.

The severity of the failure of a given resource with respect to a service can be evaluated by counting the number of versions that are still available after the failure has occurred (these are usually called redundancy degrees [10], [13]). A resource for which this number is zero is called *critical*.

B. Management of operation modes

1) Critical services: Let O_i be the set of control objectives associated with USOM m_i. As long as the services $S_i \subseteq S$ associated with m_i are available, the objectives O_i can be achieved (otherwise, the component would be inconsistently-designed). Note that this is true, by definition, whatever the available versions of these services, because the GCM does not need any specific description of *degraded performance*: if a version belongs to the list associated with a service in the GCM, it means that its performance is acceptable (whatever the way performances are defined).

When services of S_i become unavailable, some objectives of O_i might turn to be unachievable. *Critical services* are those whose unavailability implies that at least one objective of m_i cannot be achieved. The set of services $S_i \subseteq S$, associated with m_i is, therefore, decomposed into $S_i = S_i^c \cup S_i^{nc}$, where S_i^c are the critical and S_i^{nc} are the non critical ones. For example, the *regulation* service is critical for the *regulation* USOM.

2) Staying in a mode: When non-critical services become unavailable, the system can obviously remain in the current USOM, since its objectives can still be achieved: the system is fault tolerant with respect to the current USOM objectives and the current fault situation. On the contrary, when critical services of the current USOM become unavailable, its objectives can no longer be achieved, and the system is to be given other objectives. This is an objective reconfiguration strategy [7].

3) Transitions between modes: Objective reconfiguration means firing a transition towards an USOM whose objectives become the current ones. The system should, obviously, be able to achieve these new objectives, which means that in the destination USOM, no critical service is unavailable as a result of the current fault situation.

When several USOM can be reached from the current one in the USOM automaton, the choice of the destination USOM (that is, of the new system objectives) is a decision problem that must be considered at the system design stage. Unless the system objectives can be ranked according to a total ordering relation, the solution cannot be automated, thus leaving a very important role to human operators in fault situations.

4) Implementation: The proposed model has been implemented, by representing the relations between a USOM, its objectives and the services which allow the objective satisfaction with an AND-OR tree as shown on Fig. 2. Each USOM is associated with one tree, whose root is labelled by its name. The tree connects two types of nodes, namely *service* nodes and *version* nodes. The successors of a *version* node (resp. of a *service* node) are *service* nodes (resp. *version* nodes). This allows to express that a (higher level) version uses a set of (lower level) services and that a service can be provided under

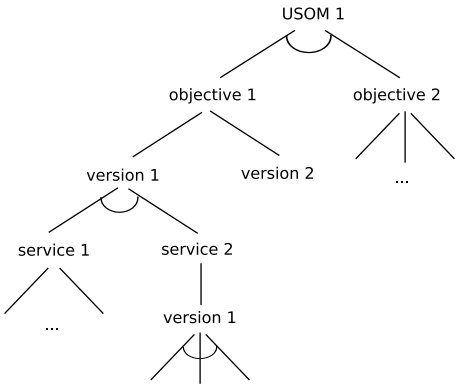

Fig. 2. AND-OR tree.

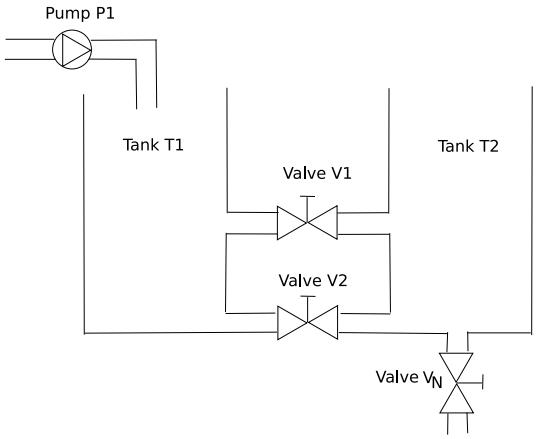

Fig. 3. Two Tank Process.

several versions. A *version* is an AND node and a *service* is an OR node. The leaves of the tree correspond to elementary services. Faults (repair operations) remove (restore) arcs in the graph. Reconfiguration possibilities result from the existence of paths between the root and the leaves. If no such path exists, USOM reconfiguration must be considered. The set of the USOM trees is analyzed at each occurrence of a fault or a repair operation, allowing operators to decide on the possibility to switch versions and stay in the current USOM or on the obligation to switch to another USOM with different objectives.

This procedure does not put a high demand on FDI algorithms, but it adapts to the information they are able to provide. In the ideal case, any single fault is known from FDI (indeed, fault detection and isolation shows which resource is faulty). Similarly, any resource repair is known. Therefore, the elementary services that are affected are directly known and the whole procedure can be run as described above. The procedure obviously accepts multiple faults: when several resources are faulty, there is a set of elementary services (instead of only one) which become unavailable. When FDI cannot isolate the fault but instead provides a set of resources amongst which one or several are faulty, then the algorithm just performs the same way: every service which needs some of the suspected resources is considered as unavailable. Finally, the reconfiguration possibilities can be researched even if FDI does not locate faults at the resource level, but at the level of non elementary services.

Following the detection and isolation of a fault, existing reconfiguration possibilities are associated with existing paths in the AND-OR tree. Finding these paths is easy, and provides the operators with a good decision support system for reconfiguration. Choosing one reconfiguration possibility, when several ones exist, is a decision problem that is not addressed here (different strategies can be used, e.g. confining the fault at the lowest possible level, switching a minimum number of components).

V. Example: the Two Tank System (TTS)

A. TTS description

The example presents a part of a level control process composed of two identical tanks (see Fig. 3). Each tank is cylindrical of cross-section area $A = 0.0154\ m^2$. Measurements available from the process are the water levels (continuous sensors L_1 for tanks T_1, and qualitative sensor L_2 for tank T_2). The qualitative values are associated with the level intervals $low = [0, 9[\ cm$, $medium = [9, 11]\ cm$ and $high =]11, 60]\ cm$.

The main aim of the TTS is to provide a continuous water flow Q_N to a consumer via an outlet valve V_N, located at the bottom of tank T_2. The water level in tank T_2 has, therefore, to be maintained at the medium level, and Tank T_1 is filled by pump P_1 up to a nominal water level of 50 cm. The flows Q_1, resp. Q_2 between the two tanks are controlled by valves V_1, resp. V_2 placed on connecting pipes at levels 0 and 30 cm. All valves can only be completely opened or completely closed. For the nominal case, valve V_2 is closed and not in use. Valve V_1 is used to control the water level in tank T_2 and tank T_1 is controlled by a PI (Proportional Integral) level controller. All cross-sections of the valves are equal to $3.6\ 10^{-5}\ m^2$.

B. TTS objectives and their organization

For the TTS, six control objectives and five USOMs are defined:

CO_0 : No action
CO_1 : Reach the level set points as quickly as possible
CO_2 : Regulate the levels to the set points
CO_3 : Completely empty the system
CO_4 : Do not spoil the environment (the liquid in the tank is supposed to be dangerous)
CO_5 : Maintain the system operation ability
$USOM_0$: No operation
$USOM_1$: Preparation
$USOM_2$: Regulation
$USOM_3$: End of production
$USOM_4$: Fall back

The USOM automaton is shown on Fig. 4 where the notation {*} specifies the control objectives the system has to achieve for the given USOM.

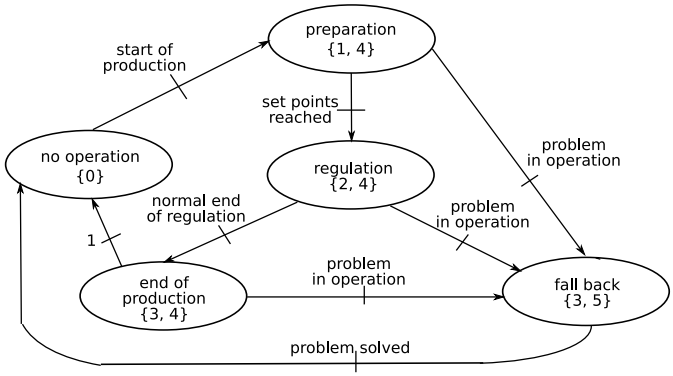

Fig. 4. TTS USOM automaton.

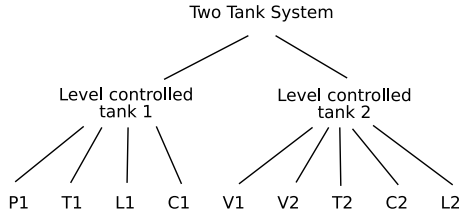

Fig. 5. TTS pyramidal decomposition.

C. TTS hierarchical decomposition

As the main objectives of the system are to keep the levels in the tanks constant, the TTS is decomposed into two subsystems, each of them associated with a level regulation loop which includes the components required to process it (Fig. 5) where C_1 is the PI controller associated with pump P_1 and C_2 is the on/off controller associated with valve V_1 or V_2.

D. Component description

The elementary components are the two valves V_1 and V_2, pump P_1, tanks T_1 and T_2, sensors L_1 and L_2 and controllers C_1 and C_2. The GCM of each of them is given below.

1) Tanks: Two USOMs can be distinguished for a tank: $USOM_0$ *(not in use)*, $USOM_1$ *(in use)*.
No objective is associated with $USOM_0$. In $USOM_1$ each tank $T_i,\ i = 1, 2$ offers the service T_i*_store* on top of the *change_USOM* service. $USOM_1$ objective is *store some quantity of liquid*. It can be fulfilled thanks to service T_i*_store*, which is described by the procedure $l_i(t) = \min\left\{\max\left\{0,\ a_i \int \Delta q_i(t)\, dt\right\},\ l_i^{\max}\right\}$ where $l_i(t)$ is the output ($prod$), corresponding to the level in tank i, $\Delta q_i(t)$ is the input ($cons$) corresponding to the difference between the input pipe and output pipe flows, and a_i is a parameter.

2) Sensors: Three USOMs are associated with the TTS sensors: $USOM_0$ *(not in use)*, $USOM_1$ *(test)* and $USOM_2$ *(automatic)*. In $USOM_1$ and $USOM_2$ sensor $L_i,\ i = 1,\ 2$ provides the service L_i*_level* on an operator request (in $USOM_1$) or on a controller request (in $USOM_2$). The procedure associated with L_i*_level* is $h_i(t) = g_i(l_i(t))$ where $l_i(t)$ is the true level in tank *i*, $h_i(t)$ is its measured value and g_i is a given function.

3) Valves: As for the sensors, three USOMs are associated with the TTS valves: $USOM_0$ *(not in use)*, $USOM_1$ *(manual)* and $USOM_2$ *(automatic)*. In $USOM_1$ and $USOM_2$, valve $V_i,\ i = 1,\ 2$ provides the services V_i*_open* and V_i*_close* associated with an operator request in $USOM_1$ and with a controller request in $USOM_2$. The associated procedure is $q_i(t) = k_i.sign\left(\Delta p_i(t)\right).\sqrt{|\Delta p_i(t)|}$ for V_i*_open* and $q_i(t) = 0, \forall \Delta p_i(t)$ for V_i*_close* where $q_i(t)$ is the flow through valve i, $\Delta p_i(t)$ is the pressure drop between its input and output, and k_i is a parameter.

4) Pump: The pump USOMs are $USOM_0$ *(not in use)* and $USOM_1$ *(in use)*. In $USOM_1$ the objective is to operate the pump according to the requests delivered by the controller using the service P_1*_deliver_Q*. As the pump is supposed to be perfect, it is described by the procedure $q(t) = Q(t)$ where $Q(t)$ is the flow parameter associated with the request for the *deliver* service, and $q(t)$ is the flow really delivered.

5) Controller C_1: Three USOMs are distinguished for C_1: $USOM_0$ *(not in use)*, $USOM_1$ *(tank_filling)*, $USOM_2$ *(level_regulation)* where the objective in $USOM_1$ is to fill up the tank as fast as possible using the service C_1*_max_flow*. The objective, in $USOM_2$ is to regulate the level in the tank using the service C_1*_regul*.

C_1*_max_flow* is associated with the procedure $Q(t) = Q^{\max}$ and C_1*_regul* is associated with the procedure $Q(t) = \min\{KP.(h_1(t) - l_1^*) + KI.\int (h_1(t) - l_1^*)dt, Q^{\max}\}$ where $Q^{\max}$ is the maximum value of the flow which can be requested from pump P_1, l_1^* is the reference level for the PI controller $(50\ cm)$, and $KP = 1.10^{-3}$, (resp. $KI = 5.10^{-6}$) are the proportional (resp. integral) coefficients of the PI regulator.

6) Controller C_2: The USOMs of controller C_2 are: $USOM_0$ *(not in use)* and $USOM_1$ *(level_regulation)* where the objective in $USOM_1$ is to calculate the valve position for level regulation using the service C_2*_calcul_pos_*V_i. This service consumes $h_2(t)$ and produces the control request for the valve V_i according the following procedure:
if $h_i(t) = low \Longrightarrow V_i = open$
if $h_i(t) = high \Longrightarrow V_i = close$

E. Aggregation procedure

1) USOM aggregation: We create the subsystem *level controlled tank 1* which aggregates pump P_1 (2 modes), level sensor L_1 (3 modes), controller C_1 (3 modes) and tank T_1 (2 modes). The product of the $USOM$ automata gives 36 compound modes, amongst which many are inconsistent, for example, (T_1*_in_use*, L_1*_test*, P_1*_not _in_use*, C_1*_level_regulation*), leaving only five consistent modes (whose label gives the functional interpretation):

LC_1*_not_in_use* = (T_1*_not_in_use*, L_1*_not_in_use*, P_1*_not_in_use*, C_1*_not_in_use*)
LC_1*_start_of_production* = (T_1*_in_use*, L_1*_automatic*, P_1*_in_use*, C_1*_tank_filling*)
LC_1*_regulation* = (T_1*_in_use*, L_1*_automatic*, P_1*_in_use*, C_1*_level_regulation*)

LC_1*_end_of_production* = (T_1*_in_use,* L_1*_automatic,* P_1*_not_in_use,* C_1*_not_in_use)*

LC_1*_maintenance* = (T_1*_not_in_use,* L_1*_test,* P_1*_not_in_use,* C_1*_not_in_use)* $\vee$ (T_1*_in_use,* L_1*_test,* P_1*_in_use,* C_1*_not_in_use)*

The $USOM$s of the subsystem *level controlled tank 2* are defined in the same way.

2) Service aggregation: Suppose the subsystem *level controlled tank 1* is in the regulation mode LC_1*_regulation*. The choice of this mode implies the choice of the modes T_1*_in_use,* L_1*_automatic,* P_1*_in_use,* C_1*_level_regulation* for the components belonging to LC_1. The low level services from which the services provided by LC_1 can be generated are:

T_1*_store* = $\langle \Delta q_i, l_1, proc_1, rqst_1, ena_1, res_1 \rangle$

L_1*_level* = $\langle h_1, l_1, proc_2, rqst_2, ena_2, res_2 \rangle$

C_1*_regul* = $\langle (h_1, w_1), Q_1, proc_3, rqst_3, ena_3, res_3 \rangle$

P_1*_deliver_Q* = $\langle Q_1, q_1, proc_4, rqst_4, ena_4, res_4 \rangle$.

The sets defined in (1)-(2) are $prod(LC_1)$=$\{l_1, Q_1, q_1\}$ and $cons(LC_1)$ = $\{\Delta q_i, h_1, w_1, Q_1\}$.

Running the sequence of low level services *measure, compute, actuate* creates a relation between h_1, w_1 and q_1. The functional interpretation is a *regulation* service provided by the high level component LC_1. The associated program is:

repeat
request the measurement service of L_1 (L_1*_level*)
request the calculation service of C_1 (C_1*_regul*)
request the actuation service of P_1 (P_1*_deliver_*Q_1)
until end of LC_1*_regul*

Since only the presence/absence of low level services (and not the way they are organized) is of interest in the sequel, the high level services are summarized by the set of low level services they need :

$$LC_1\_regul = \{T_1\_store, L_1\_level, C_1\_regul, deliver\_Q_1\}$$

In the regulation mode, services of the subsystem *level controlled tank 2* are defined from T_2*_store,* L_2*_level,* C_2*_calculate,* V_i*_open,* V_i*_close* $(i = 1, 2)$. Considering the cross-sections of the different valves and the levels in the different tanks as defined by the nominal regulation conditions ($l_1 = 50$ cm, $l_2 = 10$ cm), the outflow to the consumer is always lower than the flow through any valve V_1 or V_2. Consequently, the level in tank T_2 decreases when the two valves V_1 and V_2 are closed and increases when one valve at least is open. Two versions of the regulation service of LC_2 can then be designed by using only one valve for the regulation:

$LC_2\_regul =$
vers.1:
$\{T_2\_store,\ L_2\_level,\ C_2\_calculate,\ V_{1,2}\_close,\ V_1\_open\}$
vers.2:
$\{T_2\_store,\ L_2\_level,\ C_2\_calculate,\ V_{1,2}\_close,\ V_2\_open\}$

Using the two valves for the regulation is supposed not to be considered by the designer.

The services LC_1_regul and LC_2_regul are associated to define a service provided by the TTS itself allowing the realization of control objective CO_2. This service is defined as $sys\_regul = \{LC_1\_regul,\ LC_2\_regul\}$.

F. Fault scenarios

To illustrate the reconfiguration analysis on the TTS, consider three fault examples. In the three cases, the current operation mode is supposed to be the regulation one ($USOM_2$), and the currently used service for achieving CO_2 is *sys_regul*.

1) Scenario 1 - V_1 blocked closed: Service V_1_close cannot be stopped and service V_1_open cannot be started. Therefore, the nominal version,
vers.1:
$\{T_2\_store,\ L_2\_level,\ C_2\_calculate,\ V_{1,2}\_close,\ V_1\_open\}$
to provide LC_2_regul becomes unavailable but the degraded version,
vers.2:
$\{T_2\_store,\ L_2\_level,\ C_2\_calculate,\ V_{1,2}\_close,\ V_2\_open\}$
remains available. The service sys_regul is not directly affected by the fault and CO_2 can still be achieved, using the second version of LC_2_regul, that is using valve V_2 instead of valve V_1.

2) Scenario 2 - V_1 blocked open: Service V_1_open gets permanent in time and service V_1_close becomes unavailable. Therefore, all the versions (vers.1 and vers.2) of LC_2_regul become unavailable, which implies the unavailability of the service *sys_regul* and the impossibility to fulfill the regulation objective. The system has to be switched to an USOM in which this objective does not appear, namely the *fall back* USOM or the *end of production* USOM.

3) Scenario 3 - leak in T_1: The storage service T_1_store becomes unavailable and the environment protection objective (CO_4) can no longer be fulfilled. The system has to be switched to an USOM in which this objective does not appear, namely the *fall back* USOM. In this USOM, achieving CO_3 leads to completely empty the tanks.

VI. Conclusions

The Generic Component Model is a formalized description of the operation of devices at any hierarchical level of a controlled system. It is well suited for human operators reasoning, since the features it implements are directly connected with the operator point of view, namely operating modes, delivered services, and achieved objectives.

The GCM provides a systematic tool for finding the different reconfiguration strategies of a system when faults occur. Within a given operating mode, a system is reconfigurable if there exist different versions of the services which allow to achieve its current objectives. These versions are ranked according to a preference relation. This allows an automated real time management of the system configurations, when faults occur, as long as needed service versions are available. When the objectives of a given operating mode can no longer be achieved, the reconfiguration procedure changes the operating mode.

By automatically analyzing not only the hardware component redundancies but above all the functional redundancies (expressed in terms of services), the proposed approach takes an active part in the design of control reconfiguration laws. Unlike in multi-model approaches, all control laws must not be pre-designed, and reconfiguration solutions can be determined on line on the basis of the AND/OR tree. This tree allows to analyze the system reconfigurability in the presence of faults, since it essentially depends on the pattern of AND/OR nodes. Moreover, the approach holds for multiple faults, does not require a complete fault isolation mechanism and is not limited to a restricted class of faults. The potentially dangerous issues associated with the switching between different configurations are minimized by ranking the different versions and by favoring local reconfiguration at lower level components rather than at higher level subsystems.

As for a wide range of modelling processes, building the AND/OR tree requires human operators. Very few systems can be modelled completely automatically (except in very simple cases, for example electric circuits), and modelling paradigms which result in sets of rules to be respected (like e.g. the Bond Graphs energetic frame) are welcome. We believe that the generic component model defines such a consistent framework as far as functional modelling is concerned in a fault tolerance context.

The GCM and the associated aggregation and reasoning tools were applied to more complex systems (steam generator, complex gasifier) in the CHEM FP5 European contract[1], and they were implemented in a general Decision Support System for process supervision (see www.chem-dss.org).

REFERENCES

[1] R. J. Patton , P. M. Frank and R. N. Clark, *Fault Diagnosis in Dynamic Systems, Theory and Applications.* Englewood Cliff, NJ: Prentice-Hall, 1989.

[2] R. Iserman, "Detection based on Modelling and Estimation methods, A Survey", *Automatica*, vol. 20, pp. 387-404, 1994.

[3] M. Staroswiecki, "Quantitative and Qualitative Models for Fault Detection and Isolation," *Int. J. of Mechanical Systems and Signal Processing*, vol. 14, no. 3, pp. 301-325, 2000.

[4] T. Denoeux, "Analysis of evidence-theoretic decision rules for pattern classification," *Pattern Recognition*, vol. 30, no. 7, pp. 1095-1107, 1997.

[5] R. J. Patton, "FTC: the 1997 situation, A Survey," in *Proc. IFAC Symp. on Safety of Technical processes (Safeprocess'97)*, pp. 1033-1055, Hull, UK.

[6] J. M. Maciejowski, "Reconfigurable Control Using Constrained Optimisation," in *Proc. European Control Conference (ECC'97), Plenary Lectures and Mini-Courses*, pp. 107-130, Brussels, Belgium.

[7] M. Staroswiecki and A.-L. Gehin, "From Control to Supervision," *Annual Reviews in Control*, vol. 25, pp. 1-11, 2001.

[8] M. Blanke, M. Staroswiecki and N. E. Wu, "Concepts and Methods in Fault-tolerant Control, (Tutorial)," In *Proc. American Control Conference (ACC'01)*, pp. 2606-2620, Arlington, Virginia, USA.

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

[10] G. Hoblos, M. Staroswiecki and A. Aitouche, "Fault Tolerant Actuator Selection Using Redundancy Degrees," in *Proc. of Control'2000*, Cambridge, UK.

[11] G. Tao, S. Chen and S. M. Joshi, "An adaptive actuator failure compensation controller using output feedback," *IEEE TAC*, vol. 47, no. 3, pp. 506-511, 2002.

[12] S. Chen, G. Tao and S. M. Joshi, "On matching conditions for adaptive state tracking control of systems with actuator failures," *IEEE TAC*, vol. 47, no. 3, pp. 473-478, 2002.

[13] G. Hoblos, M. Staroswiecki and A. Aitouche, "Optimal Design of Fault Tolerant Sensor Networks," in *Proc. of the IEEE Intern. Conf. on Control Application (CCA'2000)*, Anchorage, Alaska, USA, pp. 467-472.

[14] Y. Ali and S. Narasimhan, "Sensor network design for maximizing reliability of bilinear processes," *AIChE Journal*, vol. 42, pp. 2563-2575, 1996.

[15] M. Staroswiecki and M.Bayart, "Models and Languages for the Interoperability of Smart Instruments," *Automatica*, vol. 32, no. 6, pp. 859-873, 1996.

[16] B. Chandrasekaran, J. Josephson, "Function In Device Representation," *Engineering with computers*, vol. 16, no 3-4, pp. 162-177, dec. 2000.

[17] S. Prabhakar, A. Goel , "The role of External Environment of a Device in the Functional Modelling of Engineering Devices for Adaptative Design," *Workshop Notes of Functional Modelling in Design Workshop, A.I. in Design Conf.*, Stanford Univ., June 96.

[18] L. Chittaro, R. Ranon, "Hierarchical Model-Based Diagnosis Based on Structural Abstraction," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 147-182, 2004.

[19] A. Bouras and M. Staroswiecki, "Building Distributed Architectures by the Interconnexion of Intelligent Instruments," in *Proc. IFAC INCOM'98*, Nancy, France.

[20] M. Staroswiecki, "Intelligent Sensors: A Functional View," *IEEE Tr. on Industrial Informatics*, vol. 1, no. 4, pp. 226-237, 2005.

[21] G. Gloutier, J. J. Paques, "GEMMA, the complementary tool of the Grafcet", in *Proc. of the fourth annual Canadian programmable control and automation technology conference and exhibition*, Toronto, October 12-13, 1988.

[22] E. Rakotomalala, J.-P. Elloy, P. Molinaro, D. Jampi and B. Bavoux, "Ensuring high quality in specifiactions for automotive embedded control systems," In *Int. Embedded Systems Symposium: From Specifications to Embedded Systems Applications*, Manaus, Brazil, 2005.

[23] A. Bouras and M. Staroswiecki, "How can Intelligent Instruments Interoperate in an Application Framework? A Mechanism for Taking into Account Operating Constraints," in *Proc. IFAC Symp. on Intelligent Components and Instruments for Control Applications (SICICA'97)*, pp. 465-472, Annecy, France.

[24] G. Simon, T. Kovácsházy and G. Péceli, "Transient Management in Reconfigurable Systems," *Lecture Notes in Computer Science*, vol. 1936, pp. 90-98.

[25] M. Staroswiecki, "Progressive accommodation of actuator faults in the linear quadratic control problem," *43rd IEEE Conference on Decision and Control*, Paradise Island, The Bahamas, pp. 5234-5241, 2004.

[26] X. Zhang, T. Parisini and M. Polycarpou, "Adaptive Fault-Tolerant Control of Nonlinear Uncertain Systems: An Information-Based Diagnostic Approach," *IEEE Tr. on Aut. Cont.*, vol. 49, no. 8, pp. 1259-1274, 2004.

[1]CHEM: (Advanced Decision Support System in Chemical / Petrochemical Industries) is funded by the European Community under the Competitive and Sustainable Growth programme (2001-2004) under contract GIRDCT-2001-00466

Anne-Lise Gehin was born in 1967 in France. She obtained the Engineering Degree from the Ecole Universitaire d'Ingénieurs de Lille, France in 1991. She then obtained a PhD in Automatic Control in 1994 at University of Lille, France. She has been since 1994 an Assistant Professor at Ecole Polytechnique Universitaire de Lille where she is teaching Automatic Control. She is doing her research in the Laboratoire d'Automatique Génie Informatique et Signal (LAGIS CNRS). Her research interests include Safety System Design, Fault Tolerant Control and Smart Instrumentation. She was involved in National and European projects to apply her research to machine tool, electricity, and chemical/petrochemical industries.

Marcel Staroswiecki was born in Melitopol (Ukraina) in 1945. He obtained the Engineering Degree from the Ecole Nationale Supérieure d'Ingénieurs des Arts et Métiers (silver medal), in 1968. He then obtained a PhD in Automatic Control in 1970, and the DSc in Physical Sciences in 1979, both at University of Lille, France. He joined the University of Lille as an Assistant Professor in 1969 and he became a full Professor in 1983. He is currently teaching Automatic Control at Ecole Polytechnique Universitaire de Lille and doing research at Ecole Normale Supérieure de Cachan (SATIE CNRS). He has been heading the Laboratoire d'Automatique et d'Informatique Industrielle de Lille (LAIL-CNRS), the French national network on Fault Detection and Isolation, and was a Chargé de mission at the French ministery of research. He was involved in several European projects and networks, and acts as an expert for the European Commission and the French ministery of Research. He was the IPC chair of the 2003 IFAC Safeprocess Symposium in Washington DC. He is currently working on Fault Detection and isolation, and on Fault Tolerant Control, and he co-authored two books : Actionneurs intelligents (smart actuators), Dunod, Paris, 1994 and Diagnosis and Fault Tolerant Control, Springer-Verlag, Berlin Heidelberg, 2003 (2d edition in 2006). Prof. Staroswiecki is member of two IFAC Technical Committes, Intelligent Components and Instruments, and Safeprocess.