Robotica (2003) volume 21, pp. 325–333. © 2003 Cambridge University Press
DOI: 10.1017/S0263574702004836 Printed in the United Kingdom

Smart devices for manufacturing equipment

M. Bayart

LAIL UPRESA CNRS 8021, Bât EUDIL, Cité scientifique, 59655 Villeneuve d'Ascq Cedex (France)
e-mail: bayart@univ-lille1.fr

(Received in Final Form: November 9, 2002)

SUMMARY
Smart devices used in continuous system, benefit from the addition of microelectronics and software that runs inside the device to perform control and diagnostic functions. Very small components, such as inputs/outputs blocks and overload relays, are too small to integrate data processing for technical-economic reason. However, it's possible to develop embedded intelligence and control for the smallest factory floor devices. In the paper, a generic model of smart equipment with reconfiguration functions is proposed. The interest of this functional model is that it can be used for smart devices but it can also be developed in modules for the nearest possible of the inputs and outputs in manufacturing equipment. This solution is economic for a great number of applications because it allows one to realise modular design and to standardise part of system in order to re-use it.

KEYWORDS: External model smart devices; Reconfiguration; Functional architecture; Cost effective automation.

1. INTRODUCTION
The convergence of microelectronic evolution and the increasing needs for productivity, quality and security of process for humans and environment leads to the parallel emergence of intelligent field devices, connected to fieldbuses or local area networks in real distributed automation system which insure information exchanges.

Today, such architectures are mainly implemented in continuous industrial process. The smart sensors supply is essentially relative to pressure, flow, level, . . . measure, and the currently smart actuators are valves, pumps, motor drives, . . .

However, for manufacturing installations, the sensors and actuators are, generally, less sophisticated (detector, meter, jack, . . .) than in continuous processes. A great number of automated processes (95%) require a Programmable Logic Controller equipped with logic or analogue inputs and outputs. Consequently, it's not possible to implement data storage or processing on each sensor or actuator. The alternative solution is obtained by implementing data processing units connected to some sensors or actuators and connected together by communication links in order to obtain remote inputs/outputs.

Among the manufacturing system we consider robots. We have an example of system integrated various sensors and actuators (vision system, external sensors such force, torque, . . .), in order to achieve position as well as force control. The architecture is distributed among the axes and the miniaturisation requirements prevent the integration of micro-electronic into all the sensors and actuators.

Naturally, the implementation of smart sensors and actuators in robots (or other manufacturing equipment) will allow a better reliability of information collected about the behaviour of the robot and to achieve its necessary processing information.

Moreover, it's important to the robot to have reconfiguration possibilities, especially for a reaction to disturbances that perturb the nominal functioning. One way to improve the performances of the robots is to increase the autonomy of the decisional levels in order to make quickly appropriate decisions when a disturbance occurs.

A possibility induced by the reactivity concept is to provide autonomy to low level decisional systems in order to make them more reactive to environmental fluctuations.

The actual trends are the development of smart equipment, associated to the one of fieldbuses, lead to a distributed architecture. The automation systems have been evolving from a centralised architecture to distributed one, and we obtain now automation system with intelligent distributed architecture. Robots follow this evolution, and become more and more decomposed into divided subsystems, where each of them realises an elementary function. The distributed automation systems bring several advantages, such as greatest flexibility, simplicity of the operation, best commissioning and maintenance.

Today's smart field device consists of two essential parts: A sensor or actuator module and an electronics module. The microcomputer was firstly responsible for sensor linearity, damping, p.i.d. regulation, communication, etc. Then diagnosis functions have been locally introduced, e.g. advanced diagnosis address fault detection, fault isolation[1,2] and root analysis. The early detection of anomaly, either process-related or device-related, is the key to improving plant availability and then reducing costs of production.

However, this situation can be improved by using the possibilities of local data processing to introduce reconfiguration functions.

The fault detection and diagnosis are interesting for maintenance operators. But local data processing allows one to improve the global behaviour by taking adequate decision according to the failures. The situation can generally be analysed under three major headings[3]:

- continue the system operation without a (unbearable) loss of performance;

- continue the system operation with reduced specifications;
- abandon the mission while avoiding disaster.

In this paper, we propose a functional model of intelligent equipment for distributed architectures. The advantages of this model are that it can be implemented in any hardware architecture. One possibility is to implement it in a microcontroller associated to a device (sensor or actuator) in order to obtain smart equipment. Another possibility is to integrate it in a microcomputer associated to several low cost sensors and actuators. The model rests on the notions of services and user operating modes. It can be applied to describe a system at any hierarchical level, since it allows the building of a model of aggregated components from the models of the low level ones they contain. Missions in user mode are introduced into the global system to offer reconfiguration functions.

In the first section, the concept of external model is presented; notion of services and user operating modes are introduced. In the second section, the system availability is presented according to the states of resources. As a system is a set of elementary components, which are interconnected, in order to achieve the objectives that it has been specified, in the third section notions of services and user modes of architecture of components are detailed. Then, the notion of mission is introduced to offer reconfiguration possibilities to improve the global functioning. Before conclusion, a small example illustrates the main concepts; hardware architecture to implement intelligence for small sensors and actuators used in manufacturing equipment is proposed.

2. SMART EQUIPMENT

Much work has been carried out to provide functional, behavioural, object-based, and internal or external models of smart equipment.[4,5] The external model, using the concept of service offered to users and an organisation based on operating modes, has led to a generic model description in a formal language that allows to specify and to qualify smart instruments and hybrid systems.[6]

Smart sensors and actuators integrate more and more computing power and data storage capabilities. The use of microprocessors and micro-controllers allows not only the implementation of functions that were in the past realised using analogue processing, but also the implementation of quite new ones. Internal and external models can specify smart equipment. The internal model describes the intelligent device from the point of view of the functions that it puts at work in order to contribute to the global automation system. It presents specific functions (to input, to validate, . . .) and generic ones (to communicate, to manage, . . .). The specific functions may be considered as generic for certain application classes ("to act" represents the same generic function for all the members of the specific class of electrical on/off valve).

The functional structure of a smart actuator is given on Figure 1.

The external model describes the device from the point of view of the services it is able to provide to external entities[4]

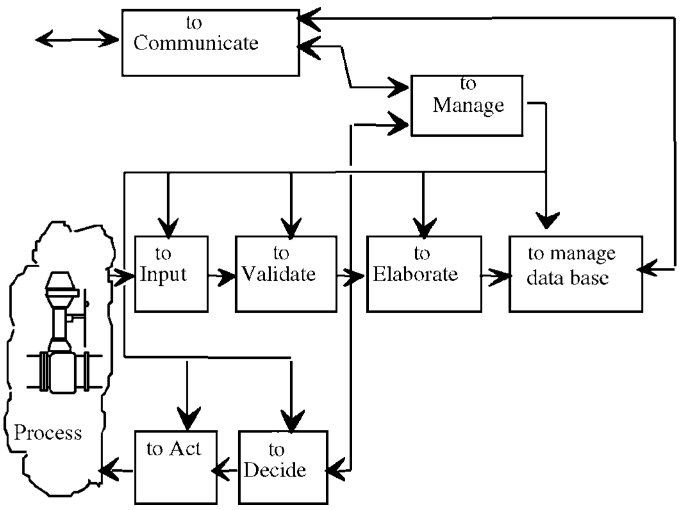

Fig. 1. Functional structure of an intelligent actuator.

(operators, other field instruments, computers, . . .). It introduces the notions of services, user-operating modes, versions of services and missions.

2.1. *Notion of service*

A **service** is defined as a procedure whose execution results in the modification of at least one datum in the device data base, or/and at least one signal on its output interface. It can be seen like a whole of functional constraints between consumed and produced variables.

Services are required by the users who intervene on the equipment during its whole life cycle, i.e. not only during its exploitation (supervision, maintenance, technical management) but through out its life cycle, from its design to its dismantling (initialisation, configuration, . . .). Two different users can be distinguished: the consumers who are the most important ones in terms of users needs, and the actors who either give information to the device or constrain it.

In order to define the obtained values, one will have to describe the computations which are done (algorithmic or sequential procedures, qualitative or fuzzy inferences, . . .), its internal configuration (parameters), the variables on which they are applied (inputs) and the required resources (hardware, software). Moreover, before it can be executed, a service must verify some activation conditions that depend on two elements the Use Condition and the Activation Request. Thus a service is described by a quintuplet (Figure 2): consumed variables (its set of consumed variables: Cs_i), produced variables (its set of produced variables: Ps_i), conditions of activation (its conditions of activation: Ca_i),

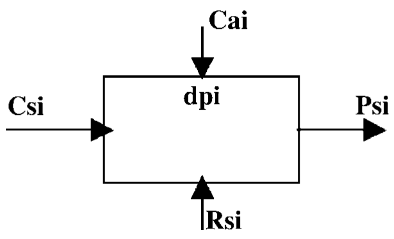

Fig. 2. Structure of a service.

data processing (dp_i), and resources (its set of resources: Rs_i).

$$s_i = < Cs_i, Ps_i, dp_i, Rs_i, Ca_i > \qquad (1)$$

Two types of components can be considered: those that integrate software and material parts and those that are only require material elements. As an example, a digital control system belongs to the first type; the main service can then be described by:

$$s_{\text{regulation}} = < \{\text{set point, position value}\}, \text{Actuator order, PID algorithm}, \{\text{energy, data unit}\}, \text{True} >$$

For a simple valve that belongs to the second one, a service can be:

$$s_{\text{regulation}} = < \varnothing, \text{value opened}, \varnothing, \{\text{energy, valve}\}, \text{presence of water} >$$

So, with these examples, in the quintuplet, the set Cs_i of consumed variables can be empty and the data processing can be non-existing. The other elements are obligatory.
The set of services, which are offered by equipment, is finite. It is called S.

$$S = \{s_i \mid i \in I\} \qquad (2)$$

I is a set of indices.

The services executions can be either dependent (precedence, mutual exclusion, . . .) or independent and concurrent. Likewise, the service can have a limited duration or can end by the occurrence of simple or complex events (operator request, emergency alarm, . . .).

The activation of a service is obtained in response to a specific request. A request is defined by:

- its name which allows to identify it;
- its execution parameters, which allow to modulate the results. The set of all the parameterised requests the intelligent instrument recognises and defines its supervisory language;
- its origin, which identifies the entity which produces it (control or maintenance operator, supervision device, control computer . . .);
- the communication link through which it is transmitted.

We define the external explicit services of equipment like services, the activation of which depends on an external event of the equipment. This type of service is obtained when the user demands its activity by a specific request.

On the other hand, we define the external implicit services like services, the activation of which depends on internal event of the equipment.

The following information for each service can be added:

- the set of resources may be distinguished by two criteria: vital/non-vital and shared/non-shared,
- the procedure in the following areas:
  - the used method,
  - priority of the service,
  - the computation time,
  - the periodicity of the service and its period if it is periodic,
  - the pre-emptability of the service.

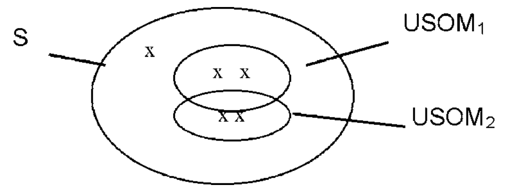

Fig. 3. USOM organisation.

2.2. *Notion of user operating mode*

A User Operating Mode (USOM) is a coherent sub-set of services. It contains at least one service, and each service belongs at least to one USOM. Moreover, in each USOM, there exists a notion of context, which allows to define the subset of services (implicit or explicit) that can be executed (implicit request), as long as the system remains in this USOM. As an example, the configuration USOM includes services of writing the various parameters, and these services are not in automatic USOM.

The union of all USOM forms a covering of the set of services of one equipment. We obtain classically the graphic representation given Figure 3.

Let MU be the set of the user operating modes.

$$\text{MU} = \{\text{mu}_j ; j \in J\} \qquad (3)$$

Each time the system is in a current USOM. The only services it will accept to run are those which belong to that USOM and, consequently, those which are coherent with the USOM objectives. In other words, any request for another service will be automatically rejected. Obviously the list of the services of each USOM has to contain a "Change USOM" request, otherwise it would be impossible to leave the current USOM. The state transition graph G(MU, T) completely specifies the nominal behaviour of a component.

Let S_{cm} be the set of change USOM services.
Let T be the set of transitions.

$$T = \{(\text{mu}_i, t_{ij}, \text{mu}_j)/\text{mu}_i \in \text{MU}, m_j \in \text{MU}, t_{ij} \in S_{cm}\} \quad (4)$$

t_{ij} indicates the logical condition required by the change from mu_i to mu_j.

Let **S** be the set of all the services that the equipment can perform.

$$\mathbf{S} = S \cup Scm \qquad (5)$$
$$\mathbf{S} = \{s_i \mid i \in I\}$$

Let Ls be the application, which associates to a USOM, the set of services, which are offered to users in this mode. **P**(S) is the set of subsets of **S**.

$$\text{Ls:} \quad \text{MU} \rightarrow \mathbf{P}(\text{S}), \qquad (6)$$

$$\text{mu}_j \rightarrow \text{Ls}(\text{mu}_j). \qquad (7)$$

We note: $S_j = \{s_i, i \in I_j\}$ with $I_j \subset I$ the set of the services which belong to the USOM mu_j.

The following properties previously stated can be written:

- $\forall j \in J$, $s_j \neq \varnothing$, a USOM contains at least one service,
- $\forall s_i \in S$, $\exists j \in J / s_i \in S_j$, each service belongs at least to one USOM (if not, the USOM would not be justified for the given application),
- $\bigcup_{j \in 1} S_j = S$, this property results from the two previous ones. The USOM set is a covering of the service set.

2.3. *Version of services*

The execution of a service requires consumed variables Cs_i and a set of hardware resources: Rs_i. Its running is then nominal if the consumed variables, which are data from the environment, are present and valid (for example: the freshness status of a transmitter is acceptable), and if the hardware resources are in good running. Unfortunately, this is not always the case, and it may happen that some of the required resources are faulty or some of the required data are not usable.

In certain cases of failure, a fault tolerant approach leads one to define specific data processing in order to use the service: its running is then degraded. In the other cases, if the service can't be achieved according to the state of hardware resources or to the characteristic of consumed variables, the service becomes unavailable.

The capacities of reconfiguration are obviously linked to the results of fault detection and isolation algorithms. For each fault on hardware component or on consumed variable that the algorithm can detect, it is necessary to analyse:

- is the service that continues to be available according to this fault in a nominal version?
- If no, is it possible to implement a degraded version that allows ensuring the service with perhaps degraded performances?
- Has the fault some effects in some degraded versions? In this case, the service can always be realised but the robustness is decreased because these versions disappear.
- The consequence of the fault is the permanent state of the device, which can be considered as the activation of a version of a service. The problem in this case is that at the time of the integration of this component in a system, it will take into account this state.

So, according to the capacities of the fault detection and isolation algorithm, the designer can envisage several versions of the same service according to the fault. In that sense, a service can be an ordered list of versions, each of them associated to a set of resources. The various versions are interchangeable; they are activated with the same request, under the same activation condition, and produce, naturally the same outputs, even if some characteristics may be different like the response time for an actuator, accuracy for a sensor. According to the state of the resources, each version of a service integrates a specific data computation.

Each version of a service is characterised by the quintuple:

$$v^k(s_i) = < C^k s_i, Ps_i, dp_i^k, R^k s_i, Ca_i > \qquad (8)$$

The following properties, i.e. determinism, consistent ordered list, have been defined in reference [9].

In order to manage the change of version, a coefficient of availability of the software and hardware resources can be introduced.

Let dr_{ik} be a function of the availability of a consumed variable or supervising resource; dr_{ik} can be defined by:

$$dr_i\colon \{C_i, R_i\} \rightarrow [0, 1] \qquad (9)$$

$r_{ik} \rightarrow$ $dr_{ik} = 1$ if the resource or the consumed variable is valid,
$dr_{ik} = 0$ if the resource or the consumed variable is not available,
$dr_{ik} = x$ if the resource is in a degraded state, or if characteristic of consumed variable is not optimal.

In fact, an infinity of versions can be defined according to the infinity of states of resources, so generally, the dr_{ik} is a discontinuous function.

Following the state dr_{ik} of every supervising resource and consumed variable of the device, services could be characterised by an indication $d(s_i)$ defined by:

$$d(s_i) = f(dr_{i1}, dr_{i2}, \ldots, dr_{ik}, \ldots, dr_{iK}) \qquad (10)$$

The whole of dr_{ik} forms a relative word to the service s_i. The availability of service is function of the resources that it uses. At a precise instant the service could be nominal, degraded, or out of order. The execution of a service depends on its availability, and therefore the formal specification of any service necessitates the description of all behaviours following its availability.

If necessary, it is possible to introduce an analogue function of availability. Then, limits have to be fixed for the selection of an adequate version, and an automaton gives the executed version of one service. For example, three thresholds $d(s_i)$ can be distinguished in order to detect the change of a version following the current version:

If $d(s_i) > \lim1$ the service is nominal,
$d(s_i) < \lim1$ the service is degraded,
$d(s_i) < \lim2$ the service is out of order.

Such an automaton is easy to present. Nevertheless, in most of cases, resources can be directly used to express a logical condition according to their availability of designer has just to establish a table of conditions of change of version on the set of service resources.

In example, let a service s_i be with its set of resources (R_{i1}, R_{i2}, R_{i3}), according to their availability: the designer can then describe a table (Table I).

Thresholds values of $d(s_i)$ correspond to logical asserts:

Nominal: $((R_{i1} = 1)$ and $(R_{i2} = 1)$ and $(R_{i3} = 1)) \leftrightarrow d(s_i) > \lim1$
Degraded: $((R_{i1} = 1)$ and $((R_{i2} = 1)$ or $(R_{i3} = 1))) \leftrightarrow \lim2 < d(s_i) < \lim1$
Out Of Order: $((R_{i1} = 0)$ or $(R_{i1} = 1)$ and $(R_{ij/(j\neq1)} = 0)) \leftrightarrow d(s_i) < \lim2$

We apply this external model with services, versions of services and user operating modes to equipment. In case of field instruments like a valve, flow sensor, . . . it's easy to implement the functional model in a micro-controller

Table I. Versions of service from the availability of the resources.

Binary availability			Version according to the $d(s_i)$ value
R_{i1}	R_{i2}	R_{i3}	
0	0	0	HS
0	0	1	HS
0	1	0	HS
0	1	1	HS
1	0	0	HS
1	1	0	DEGRADED
1	0	1	DEGRADED
1	1	1	NOMINAL

integrated with the device. In case of a small actuator or sensor, such those which are used in manufacturing and robotics, it's too expensive to associate micro-electronics to each device. In this case, embedded intelligence is associated to several actuators and sensors. The model must be extended to the system.

3. EXTERNAL MODEL OF DISTRIBUTED ARCHITECTURE

The generic model developed for smart equipment can be used for a global application. The generic model provides us with an external model, which is the user's point of view. If we consider the external services of the distributed architecture, they can be described as a reactive sequence of smart equipment and communication components, and these sequences are composed themselves by services of each equipment.[7] The knowledge of elementary equipment, their services, the versions of each service, and the global architecture allow then to define various versions of each service according to the failure of each item of equipment.

The proposed model can be applied to global architecture. A system is a definite set of interconnected discrete components. It can be built like an architecture of hardware or/and software components. An external model of application is then obtained with the aggregation of each external model of its components.

Each external service is created according to the purpose of the application. It is built from external services of its components, which, from an application point of view, become internal services. An example of an external service is given in Figure 4.

The services and the USOM can be described according to the services and USOM of each component of the global architecture.[8] A structure of internal service s_i describes constraints between consumed and produced variables. Nevertheless, functional redundancy in such a description has to be taken into account at the same time. In the example of Figure 4, s_1 and s_2 can produce the same variables that s_3 requires in order to give an average.

Fault tolerance uses redundancy to make systems more robust; such functional considerations can help designer to increase the availability of any system.

The design of the services of a global application can be realised among a bottom up approach or a top down. Let us describe quickly these two approaches.

Bottom up approach: Following a bottom up approach, we dispose of external patterns of each intelligent equipment. We are able to build global services with the services of every type of equipment. We will assume an interoperability property between every instrument on the network.

The difficulties are to build external global services by assigning services that belong to every equipment. For success, it's necessary to consider, in the quintuplet defining the concept of service, the consummate variables, and the produced variables.

A global service is built when the architect is capable to identify a reactive chain of services belonging to equipment. In practice, all reactive sequences are obtained by joining every explicit service, with the request is coming from the environment of global system, to every service, with the effect constituting the aim expected by the user (the final result of global service).

In seeking all these reactive sequences, the architect obtains some unwanted services; it doesn't constitute a problem. Indeed, the creation of global service needs to declare the implied ties of communication of each equipment towards the other equipment in the distributed architecture.

The realisation of all distributed systems necessitates the declaration of all the flows of information between the connected instruments on the network. The ties of communication must specify also their protocols of data exchange.

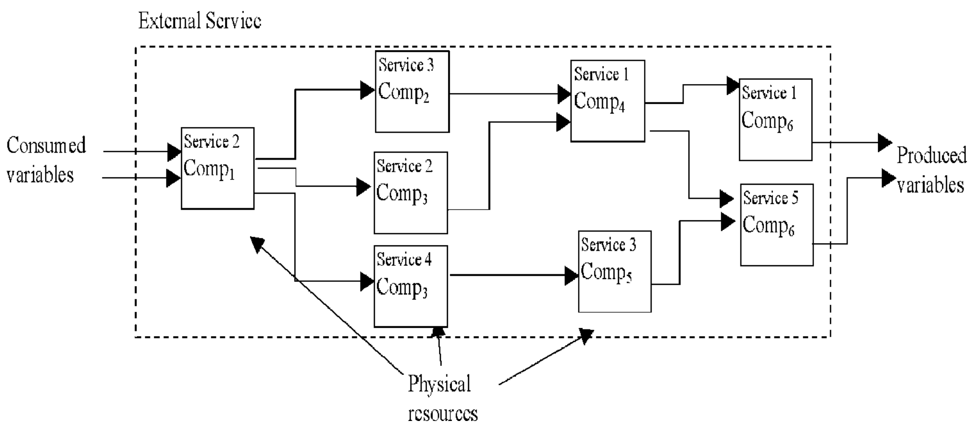

Fig. 4. Implementation of external service in the architecture of components.

The declarations of the reactive chains with USOM allow one to describe a distributed architecture.

Top down approach: It's the most employed approach in the domain of designing. If we apply the opposite reasoning to the ascending approach, we dispose of specifications. We are starting with the external pattern of the distributed application. Therefore we need to know a list of instruments that inter-operate on a network in order to realise global services of our application.

The creation of reactive chains composed by services coming from different intelligent equipment is more desirable. It consists of collecting models in a library of intelligent instruments specified according to the formalism of an external model. Starting from this library, the architect seeks some connection of the services of instruments, which permit one to find global services of the application. This research must be oriented according to different criteria:

- The respect of the operational constraints (performances of the global services, clutters problems, distances between the instruments, to take into account existing facilities),
- The number of necessary instruments,
- The cost of equipment,
- The number of access to the network for the services which risk to be executed often.

We don't observe too many subjective criteria like the name of the constructor or . . . the beauty of the designing.

3.1. *Extension of version of services*

A service of global architecture is built from services of components of this architecture. To define a version of services we have to determine software and hardware resources. In fact, requirements of global service include availability of other services (Figure 5) because they can present degraded versions.

In case of failure of one of resources of a component in a global architecture, several cases have to be considered.

- Using a degraded version can for all ensure an internal service of this component,
- If internal service can't be ensured, it is then necessary to define a degraded version of external service.

Let SI be the set of internal service required by s_i. In Figure 5, the service s_3 is in a nominal version when the required services are nominal and the consumed variables and resources are available. In the other cases, the service is in a degraded version or unavailable. The function dr_{ik} has to be modified. dr_{ik} can be defined by:

$$dr_i: \{d(s_i)/s_i \in SI\} \cup \{Cs_i, Rs_i\} \rightarrow [0, 1] \qquad (11)$$

$r_{ik} \rightarrow dr_{ik} = 1$ if the resources or the consumed variable is valid and if $\{d(s_i) = 1 \forall s_i \in SI\}$.

$r_{ik} \rightarrow dr_{ik} = 0$ if the resources or the consumed variable is unavailable, or if $\{d(s_i) = 0 \forall s_i \in SI\}$.

$r_{ik} \rightarrow dr_{ik} = \alpha$ in the other cases.

When $dr_{ik} = \alpha$, an analysis of S must be realised in order to determine versions of all external services (nominal, degraded or unavailable).

A success diagram can be built from the causal links between consumed and produced variables. This diagram gives the fault propagation phenomenon[9] and permits to deduce effects (Figure 6).

The robustness of services (s_i), providing by versions of the services, allows a naturally improving robustness of global services that influence system availability. In Figure 6, the global service availability depends directly on $d(s_5)$. The internal service s_3 is required by s_2 or s_1. Any failure on s_2 doesn't make the external service unavailable because s_3 remains if s_1 is right. On the other hand, any default on s_4 makes S′ unavailable.

$$d(s_5) = f(d(s_4, d(s_3), dr_{5l}, \ldots, dr_{5K}) \qquad (12)$$
$$with\ d(s_3) = f(d(s_2 \vee d(s_1)), dr_{3l}, \ldots, dr_{3L})$$

Following the availability of each internal service, an analysis must be made on several versions of the global service. Table II illustrates it.

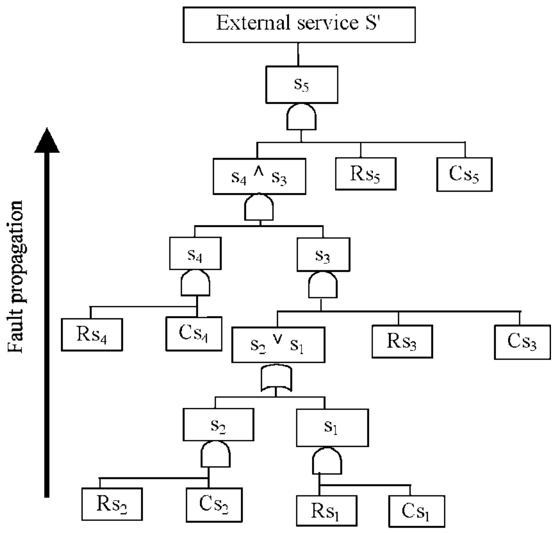

Fig. 6. Success diagram of external service.

Table II. Availability study of S′.

$d(s_1)$	$d(s_2)$	$d(s_3)$	$d(s_4)$	$d(s_5)$	$d(global\ service)$
1	1	1	1	1	1
1	1	1	1	0	0
1	1	1	0	1	0
1	1	0	1	0	0
α	1	β	1	δ	δ
. . .	. . .	. . .	. . .	. . .	$d(s_5)$
0	0	0	1	0	0

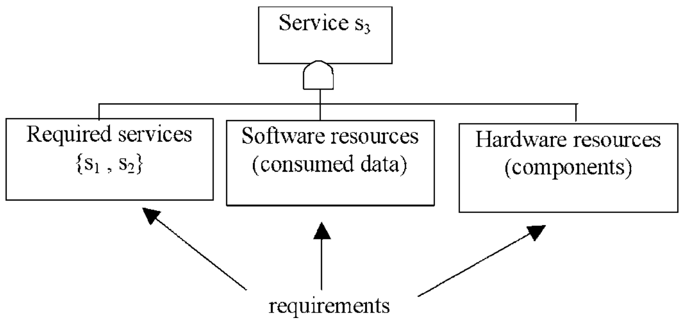
Service s_3
Required services {s_1 , s_2}
Software resources (consumed data)
Hardware resources (components)
requirements

Fig. 5. Example of requirements of service.

The changes of version constitute the most delicate part because they have to indicate the version source and the version destination of the described transition, and they have to express clearly the whole of logical conditions permitting the change of version. The conditions of change of version are generally explained with assertions, as it has been explained for a component.

At the system level, the user operating mode can be applied in the same manner as in the component. However, it can be interesting to introduce another level: the missions (ME), as introduced in reference [10]. The missions allow distinguishing various situations at different times that correspond to different objectives to perform in a same user mode, for example, to reach the defined set point, to maintain this fixed position, etc.

The various missions correspond to various aggregations of elementary services. One feature of them is to take into account the unavailability of elementary services, and to consider the behaviour of the system in case of failure. In that sense, different possibilities have to be defined to ensure external service in spite of faults in certain components by finding various ways inside this structure to elaborate produced variables.

3.2. *Notion of missions*

Robust external services can be defined as a structure, which presents more than one manner to accomplish a mission of this service. In that way, one can demonstrate divers possibilities inside such structure with a functional diagram of transformations (values and type of values).

The various ways inside a structure provide external service (Figure 7). However, what's the executed way in nominal cases? A preference has to be specified for one way. Criteria to choose this way can be indicated:

- the number of components to produce the service,
- the time to execute the service, (can the longest service be considered as the degraded version of the same entity? (If a fault occurs, and if the external service changes, a hypothesis has to be made. Nevertheless, the new service with a different configuration has a nominal version. The quality of the service is the same, and yet performances are different.),
- the number of connections across the network equipment.

Finally, a graph can be used to represent resources which are required in the various ways to accomplish the mission of an external service.

At the system level, versions of services depend on the availability of the hardware resources, which is evaluated by fault detection and isolation algorithms, like for elementary equipment. However, global vision allows one to take into account the appearance of a fault of one elementary component which produces the activation permanently in time of a specific service, for example, if a relay has jammed in an open position, the closed position service becomes unavailable while the opening service is always active. In this case, it's necessary to adapt the service of the other components to achieve a global service of the subsystem.

It's at that level, that the notion of missions assumes its importance. When a fault occurs, several cases have to be considered.

It's possible to ensure the wished services, because there exists at least one version that allows its execution, and the external service is realised with another version.

It is not possible to find a degraded version to realise the service; it has to be stopped. The service becomes unavailable and the mission that requires this service becomes unavailable.

However, if the designer could foresee this type of fault (internal service permanently active), he could define another mission which leads, for example, to a safety situation. In other words, the advantages of assembling several devices allow not only to improve the efficiency but also to define several missions depending of failures.

The reconfiguration algorithm evaluates the availability of each internal service from the list of faulty resources provided by FDI Algorithms; this allows one to calculate the availability status of the external service. Then according, to the availability status of this service, the possibility or not to keep one of the missions is evaluated.

3.3. *Example*

Let's consider an example of a system of regulation (Figure 8). There are different options to build such system: the

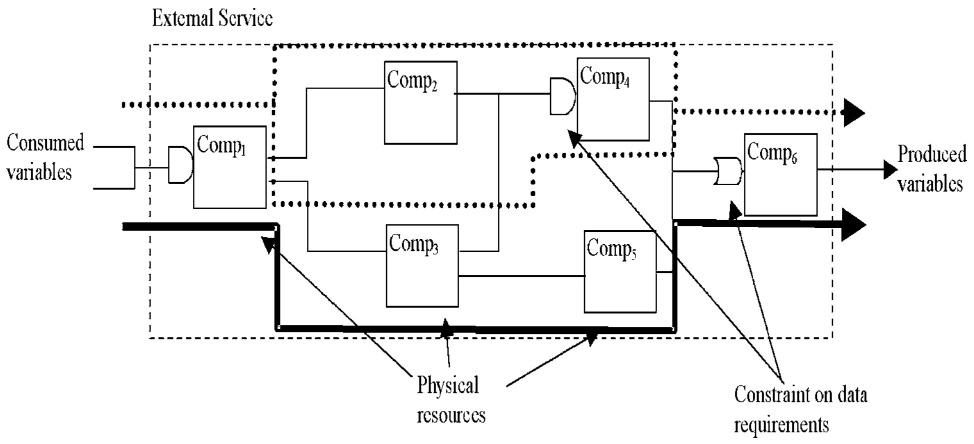

Fig. 7. Various ways for a service execution.

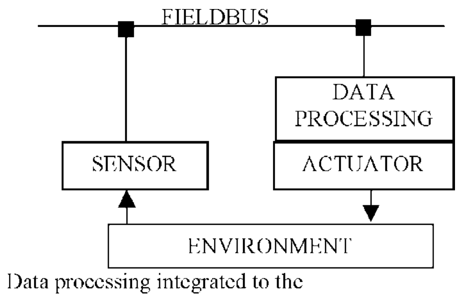

Fig. 8. A regulation system.

actuator is associated with a processor or the data processing unit alone . . . Usually, this system can be used in closed loop with a sensor for reactivity of the application.

The regulation system will be considered in three USOMs: Idle, Configuration and Automatic.

With the identified components in the "Automatic" mode, the external service "to Regulate" (Figure 9) is easily built for a closed loop configuration. This service uses the following components:

- Smart sensor,
- Smart actuator (actuator and processor),
- A subset of components of network equipment.

The global communication system is taken into account as equipment. It offers a service of communication Scom1.

The service "to Regulate", in a closed loop, can be identified like the composition of the service "to Measure", the service Scom1, the service "to elaborate command", and the service "to Act".

The service "to measure" offers only a nominal version. The service scom1 can have degraded versions of there is a redundancy of the fieldbuses. The service "to elaborate command" has a nominal version and a degraded version that leads to emit a specific command.

The service "to act" has a nominal version in an automatic mode, but the service "to act manual" is offered in a manual mode.

The main problems which can appear are due to:

- a failure of the sensor that leads to the sending of the same value.
- A failure of the communication system which becomes out of order.
- A failure of the actuator which stays on a blocked position.

In the first two cases, the external service "to regulate" becomes unavailable. Another mission can be implemented by permitting to have an open loop control. In that sense, in

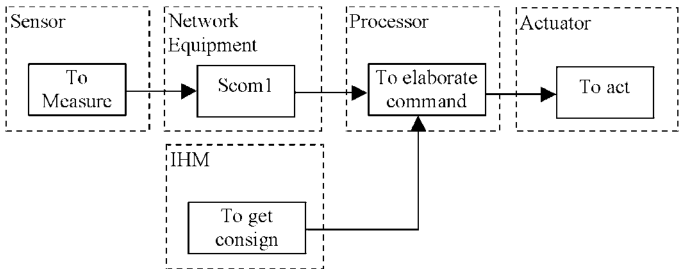

Fig. 9. Regulation service composition.

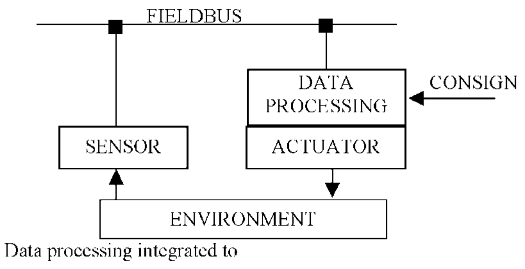

Fig. 10. A regulation system with local control.

the design of the system, an interface IHM can be implemented to give an order to the actuator (Figure 10).

This hardware solution allows one to define several ways and thus several missions to obtain a regulation of the physical process.

Also, according to the set of hardware resources, it is possible to elaborate a management of external services of the global architecture according to the availability of internal services.

In case of failure of the actuator, and, in particular, a blocked position, the two missions become unavailable, and a maintenance operation is required.

This small example shows that in the data processing unit it's possible to implement the external model with some reconfiguration functions.

Small devices for which it's not possible to integrate micro-electronics for techno-economic reasons, it's possible to implement the proposed model in a micro-controller embedded near the sensors and actuators.

4. HARDWARE ARCHITECTURE

The external model defines the functional architecture, but doesn't specify the operational architecture. In case of small sensors or actuators, the proposed hardware architecture to implement the model is given in Figure 11. It integrates three parts:

- a communication unit which allows the exchange with the other data processing units (PLC or PC). (Various fieldbus, such as Profibus FMS/DP, Can Open and Device Net can be used.)
- A data processing unit which can be connected to a human/machine Interface or programming console.
- 1 to n analogue or logic Input/output modules for which the users can connect sensors and actuators.

The aim of such an architecture is to develop generic components, which present a lot of advantages, techno-economically speaking in the various phases of life.

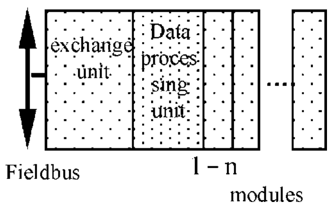

Fig. 11. Hardware Architecture for multiple small sensors and actuators.

At the setting up of the smart sensors or actuators, the use of generic modules permits one to configure easily the software according to the needs of application, and eventually load necessary functions and parameters from the library. This possibility offers flexibility for the application and an additional capacity to adapt components to the needs, even in case of evolution along their life. Another aim is following on the possibility to easily test subsets of the application before setting up on a definitive site; the acceptance trials will then be simplified and reduced. The decreasing of wiring due to the use of fieldbuses allows one to save twenty to thirty percent of the global costs of automation.

During the exploitation of the equipment, the embedded data processing reduces the cycle time, and the adaptability allows quick reconfigurations for the production changes. A part of the automation can be stopped when another one is used.

In the maintenance stage, the diagnosis functions that can be locally integrated allow one to detect quickly and to locate the origin of a failure. Then, the diagnosis information is clearly indicated to the operator on his interface and the failure can be immediately corrected. In addition, the diagnosis information can trigger off a stop of an operation.

The modular design can be disconnected during the maintenance intervention, or put the connected devices in a retiring position. We can add that the use of generic modules for hardware architecture limits the inventory. The last advantage of this modular architecture is to manage easily the developments, because they concern essentially the software.

5. CONCLUSION

In this paper we considered the generic models, which have been developed to construct smart equipment. It is based on the notion of services and user modes that are detailed in the first part. This can be applied for an isolated component (sensor or actuator) or a global system. The presence of Fault Detection and Isolation Algorithms allows one to improve the safety of the equipment we show how Fault Tolerant Control aspects can be introduced by versions of services.

The external model can be applied easily for smart sensors and actuators. However, it isn't an economical solution for small devices. In that sense, we propose, in the second part, to extend the external model to a system by a composition of services. The aim is to improve the global functioning of several devices, as well as in terms of productivity, than in terms of global safety.

The existence of several versions that provide the same service allows systems reconfiguration and increases the system fault tolerance, since the unavailability of a given resource does not systematically imply the unavailability of the service, which uses it. Thus, the existence of several missions for the system allows one to implement fault tolerant control, i.e. to develop a decision system which allows to continue some actions according to the failures and the possibility of redundancies.

The external model can be implemented in a centralised computer, but it can also be implemented in a micro-controller in order to realise the embedded intelligence for several devices.

The advantage of such modules is to implement data processing as near as possible to the inputs and outputs. This solution is economical for a great number of applications because it allows one to realise modular design and to standardise parts of systems in order to re-use them. The decentralised architecture offers easy commissioning and maintenance. At last, it gives some intelligence to sensors and actuators for which integration of data processing is not justified for technical-economic reasons.

References

1. R.J. Patton, P.M. Frank and R.N. Clark, *Fault Diagnosis in Dynamic systems – Theory and applications* (Prentice-Hall, Englewood Cliff, N.J., 1989).
2. R. Iserman, “Detection based on Modelling and Estimation methods, A Survey”, *Automatica* **20**, 387–404 (1994).
3. J.M. Maciejowski, “Reconfigurable Control Using Constrained Optimisation”, *ECC’97*, Brussels, Belgium, Plenary Lectures and Mini-Courses (1997) pp. 107–130.
4. M. Robert, M. Marchandiaux and M. Porte, *Capteurs Intelligents et Méthodologie d’Evaluation* (Hermès, Paris, 1993).
5. M. Staroswiecki. and M. Bayart, “Models and Languages for the Interoperability of Smart Instruments”, *Automatica* **32**, No. 6, 859–873 (1996).
6. M. Bayart, E. Lemaire, M.A. Peraldi and C. Andre, “External model and SyncCharts Description of an Automotive Cruise Control System”, *Control Engineering Practice* **7**(10), 1259–1267 (Oct., 1999).
7. J.J. Kenney “Executable formal models for distributed transaction systems based on event processing”, *PhD thesis* (University of Stanford, June 1996).
8. C. Choukair and M. Bayart, “Application of external model of intelligent equipment to distributed architectures”, *ISAS’99, 5th International Conference on Information Systems Analysis and Synthesis* Orlando (1999), **Vol. 4**, 329–335.
9. J.C. Laprie, “Sûreté de fonctionnement des systèmes informatiques et tolérance aux fautes: concept de base”, *TSI* **4**, No. 5, 419–429 (1985).
10. A.L. Gehin, M. Staroswiecki and M.L. Assas “A Bottom-up Approach to Analyse Reconfiguration Possibilities”, *Workshop on Principles of Diagnosis*, Loch Awe, Scotland (1999) pp. 100–108.