On the formalisation of integrating watchdogs into discrete event controller structures

Gábor Kovács*, Laurent Piétrac†, Bálint Kiss*, Eric Niel†
* Department of Control Engineering and Information Technology
Budapest University of Technology and Economics, Budapest, Hungary
† Laboratoire Ampère, INSA-Lyon, Villeurbanne, France

***Abstract*— This paper reports a low-cost online fault detection approach for supervisory controllers in the framework of Supervisory Control Theory (SCT). For the cases when sensors dedicated to fault detection increase significantly the cost of controllers, or failure events are even impossible to detect by a direct way, methods based on the well-known watchdog structures are proposed. To successfully integrate watchdogs in the SCT framework, their discrete-event model is defined, and fault-detection techniques proposed in this paper are based on the extension of controller models previously designed using conventional supervisory synthesis methods. Fault-detection strategies are presented for centralized and distributed supervisory control environments, in the latter case providing solutions for avoiding problems according to fault propagation. Proposed techniques give full authority to the system designer in defining failure handling procedures and are proved not to influence the operation of the processes when no fault occurs. Since the extension of the controller models is defined by a formal and systematic manner, suitable algorithms based on the presented techniques can be constructed to allow automatic integration of fault-detection capabilities into existing controller structures.**

I. Introduction

The need for dependable and fault-tolerant systems has arisen in the last decades. In application areas like automotive and aerospace industry, nuclear technology etc. reliability and safety is a key issue and these properties have to be fulfilled regardless to the cost. However, the need for dependability has also arisen in other industrial or even consumer electronics products, where financial reasons or restrictions on the on-market-time limit the use of fault detection and fault diagnostic technologies. [1]

The theory of discrete event systems provides a suitable framework for the design of supervisory control structures, which may be responsible for assuring the safe operation of sophisticated or large-scale systems. [2] Several propositions have been presented in the field of fault detection, failure identification and failure diagnosis for discrete event systems, see for example, [3], [4], [5], [6]. Although these methods provide general and theoretically based solutions, they are mainly not applicable in everyday practice due to their high computational needs.

This paper reports a practice-oriented, low-cost online fault detection approach using the well-known watchdog structures. Although watchdogs are used for decades to monitor even hardly observable failures, their models and the related fault detection methods have not yet been formalized in the discrete event framework. Nevertheless, system engineers often integrate watchdog based fault detection solutions into their controller structures, but they can use only ad-hoc, informal, and therefore undependable design methods.

We introduce the discrete-event models of the watchdog to allow their integration into the framework of Supervisory Control Theory (SCT). Some formalized and systematic procedures will be presented that allow the extension of existing controller models in order to implement watchdog-based fault detection strategies in centralized and distributed control environments. Due to their formal and systematic properties, these procedures can be easily automated by suitable algorithms to help the integration of fault-detection algorithms. Moreover, the presented methods do not restrict the failure handling procedures, so they allow high flexibility for the system designer.

Structures and methods presented in this paper are not restricted to the formal framework of SCT. In industrial practice, supervisory controller structures are often designed in an informal, intuitive way using finite state machine-based modeling tools (e.g. Grafects). Although the design procedure excludes the use of formal methods, the proposed fault-detection methods and structures can be used, and by their automatic integration the developement cycle can be accelerated. Moreover, since these controllers are often tested on simulated or physical processes to check if they meet the proposed requirements, watchdogs can be also used to guard the plant against controller design-related malfunctions (e.g. infinite cycles) during the prototyping phase.

The remaining part of the paper is organized as follows. Section II gives a short overview on SCT and on the supervisory control design procedure. In Section III the principles of watchdog-based fault detection techniques will be introduced. Sections IV and V present the proposed fault-detection strategies for centralized and distributed control architectures, respectively. Section VI concludes the paper.

II. Preliminaries

We present here only some fundamental principles and notations of SCT in order to keep the paper a selfcontained as possible. For more details, the reader may refer to [7].

The discrete-event system G is described by the 5-tuple $G = \{Q^G, \Sigma^G, \rho^G, q_0^G, Q_M^G\}$ with Q^G as its state set, Σ^G as its event set, $\rho^G : Q^G \times \Sigma^G \rightarrow Q^G$ as its partial transition function, q_0^G as its initial state and Q_M^G as the set of its

marking states. The event set Σ^G can be divided into the distinct sets of controllable and uncontrollable events so that $\Sigma^G = \Sigma^G_C \cup \Sigma^G_U$ where $\Sigma^G_C \cap \Sigma^G_U = \emptyset$. The notation $\exists \rho(q, \sigma)$ means that there exists a transition associated with the event $\sigma \in \Sigma^G$ leaving the state $q \in Q^G$. The language generated by G is denoted by $L(G)$. The constraints to be respected by the supervised system are given by the specification modeled by an automaton denoted by E.

The goal of supervisory synthesis is to define a supervisor which can restrict the operation of the system to meet the constraints of the specifications, so that the supervisor S is a function $S : L(G) \rightarrow \Gamma$ defined by $\Gamma = \{\gamma = PWR(\Sigma) \mid \gamma \supseteq \Sigma_U\}$ where γ represents the set of events authorised by S and $PWR(\Sigma)$ is the set of all subsets (the power set) of Σ. If the specifications are controllable, the automaton S/G describing the supervised system is the product of G and E. In other cases the maximal permissive sublanguage can be found, which allows the greatest possible set of controllable events, see [8] and [9].

The controller model is also described by a 5-tuple $C = \{Q, \Sigma, \rho, q_0, Q_M\}$, possibly extended by a control map $\Theta : Q \times \Sigma_C \rightarrow \{0, 1\}$. The controller C is constructed based on the automaton representing the supervised system and the supervisor itself, and gives the automaton model of the controller with the events to be enabled or disabled in each of its states defined by the control map.

Finite state machines are illustrated by transition diagrams in this paper (see, for example, Fig. 1). An arrow entering a particular state denotes the initial state, while an arrow leaving a particular state but not leading to any other state denotes a marking state. A tick on an arrow represents that the event associated to the given transition is controllable.

Formal methods of supervisory controller synthesis lie on the principles above, but are placed in a more complex, general control design framework. We should consider that the objective of the design process is to realize a controller, i.e. implement it using a suitable hardware platform, ensuring that the supervised process meets the proposed requirements.

Controller design is based on the models of the process and the requirements, given commonly in the form of finite state machines. The supervisor is synthesized upon these models, and then a controller model, which is a representation of the supervised system, is derived. The controller model is then implemented on a suitable platform, and it is tested along the process to verify whether the supervised system meets the requirements.

Our aim is to present methods to implement watchdog-based fault detection techniques by extending previously designed controller models. In this paper, we shall assume that the controller model has been previously designed and described by an FSM.

III. Principles of watchdog-based fault detection

Watchdog structures are commonly used for fault detection in electronic controller devices (see, for example, [10]). Watchdogs are used to signal if a given operation (e.g. displacement of a workpiece) is not finished in a predefined time period, therefore the presence of a failure can be assumed.

The operation of the watchdog can be pictured as follows. The watchdog is started by the controller before executing a given task. Then, it starts counting and when reaches a predefined final value, it outputs an alarm signal. If the controller resets the watchdog before reaching the final value, the alarm signal is not generated.

Among their simplicity, a main advantage of watchdogs compared to other fault-detection techniques is that they do not need dedicated sensors for unrevealing the faults. For example, using a watchdog the failure of a conveyor line can be assumed if no workpiece arrives at a given location in a time period, so no additional sensor measuring the speed of the conveyor or the force of the motor driving it should be added, and therefore the cost of the controller implementation can be reduced.

Depending on the architecture, watchdogs can be implemented as software routines or hardware components. The latter one, used in fault-critical applications, can be composed of simple logical components, such as a counter, a memory block for storing the final value, a comparator, and an alarm logic to maintain the alarm signal after reaching the final value.

In order to define formal methods for implementing watchdog-based fault detection in the SCT framework, at first some notations should be clarified.

Roughly speaking, a task is a part of the supervised operation of the plant, which can be clearly distinguished from other activities. A task is started by a suitable, controllable event, and its successful completion is indicated by one or more confirmation events. For example, a task can be the displacement of a workpiece by a robot arm. In this case, the task is started by downloading the new coordinates to the controller of the robot arm, and the successful completion can be indicated by a sensor at the target position. Notice that the confirmation event is not necessarily generated by the subsystem the given task is implemented in.

Definition 1: A task $T_i = \{Q^T_i, \Sigma^T_i, \rho^T_i, q^T_{i,0}, Q^T_{i,M}\}$ is a suitably chosen subsystem of an existing controller model: $T_i \subseteq C$, with $Q^T_i \subseteq Q$, $\Sigma^T_i \subseteq \Sigma$, $q^T_{i,0} \in Q^T_i$, $Q^T_{i,M} \subset Q^T_i$. The task is a set of continuous trajectories, with $q^T_{i,0}$ as its first state, so that $\forall q \in Q^T_i : \exists t \in {\Sigma^T_i}^*, \rho(q^T_{i,0}, t) = q$. The last states of the trajectory are in the set $Q^T_{i,M}$, so that $\forall q_{i,M} \in Q^T_{i,M}, \forall t \in \Sigma^* : \rho(q_{i,M}, t) \notin Q^T_i$.

Definition 2: The set of the tasks associated with a system will be denoted by $T = \{T_1, T_2, \ldots, T_n\}$, where n is the number of the tasks associated to the given controller model. It is assumed that the tasks are not overlapping each other, so $\rho^T_i \cap \rho^T_j = \emptyset \ \forall i, j \leq n, i \neq j$.

Definition 3: If $\exists_1 \sigma^* \in \Sigma_{i,C}$ so that $\exists \rho(q_{i,0}, \sigma)$ if and only if $\sigma = \sigma^*$, then the controllable event σ^* will be referred as the command event of the task T_i and will be denoted by σ^{CMD}_i.[1]

[1] $\exists_1$ stands for 'there exists a unique'

Definition 4: The events indicating the successful completion of the task T_i will be denoted by $\sigma_{i,j}^{CONF} \in \Sigma_i^T$ in the sequel and will be collected to the set of confirmation events $\Sigma_i^{CONF} = \{\sigma_{i,1}^{CONF}, \sigma_{i,2}^{CONF}, \dots \sigma_{i,n}^{CONF}\}$.

Remark: The selection of confirmation events is an intuitive task of the system designer.

Definition 5: The task T_i is said to be possible to put under the guard of the watchdog if and only if

1) $\exists \sigma_i^{CMD}$ and
2) $\exists \rho(q, \sigma), q \in Q_i^T, \sigma \in \Sigma_i^{CONF} \Leftrightarrow \rho(q, \sigma) \in Q_{i,M}^T$.

Remark: In the followings, all tasks should be assumed to be possible to put under the guard of the watchdog.

Faults are handled by the so-called alarm handling procedures, which are executed upon an alarm signaled by the watchdog. However their definition is left entirely to the system designer, providing full flexibility for their realization, some assumptions have to be made according to them.

Depending the nature of the failure, i.e. during which task it has occured, various alarm handling procedures can be defined. However, it is possible that the same failure handling is required for different tasks, e.g. the intervention of a human operator is needed in several cases. In order to allow watchdogs to start alarm handling procedures according to the given task, the first state of each alarm handling procedure and their association to the tasks should be clearly defined.

Definition 6: The alarm handling state $q_{AH,i}$ is the first state of the i^{th} alarm handling procedure of the actual controller model. Alarm handling states are collected to the set $Q_{AH} = \{q_{AH,1}, q_{AH,2} \dots q_{AH,n}\}$.

Definition 7: The function $\xi : T \rightarrow Q_{AH}$ realize the association between tasks and alarm handling procedures, so that the alarm handling state associated with the task T_i is $q = \xi(T_i), q \in Q_{AH}$

The integration of watchdogs into existing supervisory control structures is built up from three main steps, which have to be carried out for all the tasks.

At first, the task to be put under the guard of the watchdog should be selected. An ideal candidate can be clearly distinguished from other activities of the controller, i.e. it has well-defined command and confirmation events.

The second step is the definition of the alarm handling procedure according to the given task. It should be designed intuitively by the system designer, and should ensure safe operation or the execution of an emergency shutdown. It is recommended, however not compulsory, to reset the watchdog at the end of the alarm handling procedure.

The third step is the extension of the controller model(s) in order to incorporate watchdog-based fault detection capabilities. The methods of extension will be discussed in the following sections, where it will be assumed that the controller models are already containing the alarm handling procedures.

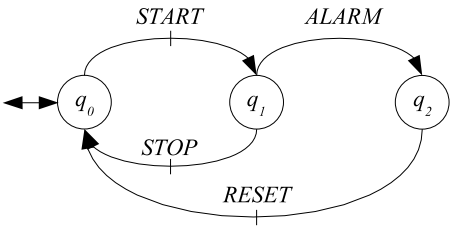

Fig. 1. Discrete-event model of the watchdog

IV. Fault detection in centralized control environment

A. Discrete event model of the watchdog

To implement watchdog-based fault detection methods in the SCT framework, at first the discrete-event model of the watchdog should be defined. The operation of the watchdog in a discrete event framework can be captured as follows. It can operate in three states, namely $Idle$ (q_0), where the counter, left unmodeled at this level of abstraction, is stopped, $Running$ (q_1), where the counter is running but the final value has not yet reached, and $Alarm$ (q_2), where the alarm signal is issued. The transitions between these three states can be associated to the events of starting the watchdog (START), stopping it (STOP), the issue of the alarm event when reaching the final value (ALARM) and the reset of the watchdog (RESET). All of these events are controllable, except the ALARM event, which is generated by the watchdog itself. The discrete-event model of the watchdog is given by Fig. 1, while its evolution is illustrated in Fig. 2.

B. Extension of the controller model

In order to implement watchdog-based fault detection methods in existing supervisory control structures, the model of the controller should be extended. In the followings, it shall be assumed that failure handling procedures have been already defined and integrated to the controller model by the

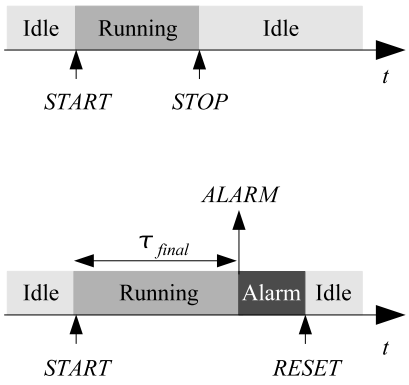

Fig. 2. Evolution of the watchdog

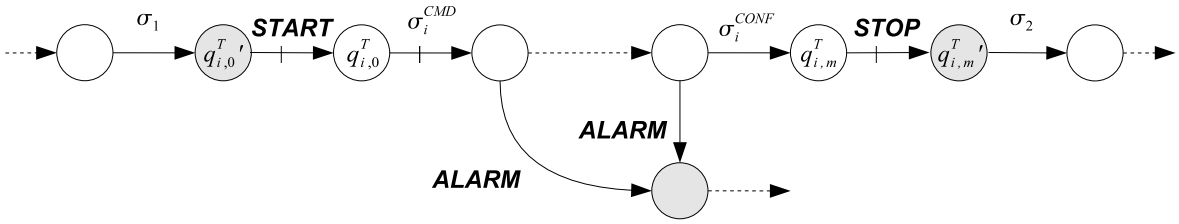

Fig. 3. Extension of the controller model in centralized environment

system designer, as well as their associations to the tasks to be put under the guard of the watchdog.

The controller model is initially described by a 5-tuple $C = \{Q, \Sigma, \rho, q_0, Q_M\}$, possibly extended by a control map $\Theta : Q \times \Sigma_C \rightarrow \{0,1\}$, which will be modified, resulting in extended model(s), $C' = \{Q', \Sigma', \rho', q_0', Q_M'\}$ and, if Θ exists, $\Theta' : Q' \times \Sigma_C' \rightarrow \{0,1\}$. The aim of the extension is to put the task $T_i = \{Q_i^T, \Sigma_i^T, \rho_i^T, q_{i,0}^T, Q_{i,M}^T\}$ under the guard of the watchdog. To do so, the watchdog should be started before executing the task, and stopped after its successful completion. The handling of alarm events should also be guaranteed by starting the appropriate alarm handling procedure. The extension is defined formally by the followings.

A new state associated to $q_{i,0}^T$, namely $q_{i,0}^T{}'$ and new states associated to each state $q_{i,j}^T \in Q_{i,M}^T$, namely $q_{i,j}^T{}' \in Q_{i,M}^T{}'$ so that $|Q_{i,m}^T| = |Q_{i,M}^T{}'|$ should be added to the state set of the controller[2]: $Q' = Q \cup \{q_{i,0}{}'\} \cup Q_{i,M}^T{}'$. The events of the watchdog should also be added to the event set of the controller model: $\Sigma' = \Sigma \cup \{\text{START}, \text{STOP}, \text{RESET}, \text{ALARM}\}$. Then, the partial transition function of the controller should be extended to $\rho'(q,\sigma)$ for $\forall q \in Q'$ and $\forall \sigma \in \Sigma'$ by the followings.

$$\rho'(q,\sigma) =$$

$$\begin{cases} \rho(q,\sigma) & \forall q \in Q \setminus Q_{i,M}^T, \forall \sigma \in \Sigma \\ \rho(q_{i,j}^T,\sigma) & \forall q_{i,j}^T{}' \in Q_M', \forall \sigma \in \Sigma \\ q_{i,j}^T{}' & \forall q_{i,j}^T \in Q_M, \forall \sigma = \text{STOP} \\ \xi(T_i) & \forall q \in Q_i^T{}' \setminus Q_{i,M}^T{}' \setminus q_{i,0}^T, \sigma = \text{ALARM} \\ q_{i,0}^T{}' & \text{iff } \rho(q,\sigma) = q_{i,0}^T \\ q_{i,0}^T & \text{iff } q = q_{i,0}{}', \sigma = \text{START} \\ \text{undefined otherwise} \end{cases}$$

The extension of the controller model is illustrated in Fig. 3. The modification of the control map is shown by Table I.

Proposition 1: If no failure is signaled by the watchdog, i.e. no ALARM event is generated, the plant G acts the same under the supervision of the original and the extended controller by the mean that it generates the same language, so that $P_{\Sigma^G}(L(S'/G')) = L(S/G)$.

[2] $|Q|$ denotes the cardinality of the set Q

TABLE I

MODIFICATION OF THE CONTROL MAP

	START	STOP	σ_i^{CMD}	Σ_{REST}[1]
$q_{i,0}^T$	0	0	1	$*$[2]
$q_{i,0}^T{}'$	1	0	0	$\Theta(q_{i,0}^T, \sigma)$
$q_{i,j}^T \in Q_{i,M}^T$	0	1	0	$\begin{cases} 0 \text{ if } \exists \rho(q_{i,j}^T, \sigma) \\ \Theta(q_{i,j}^T, \sigma) \text{ ow}[3] \end{cases}$
$q_{i,j}^T{}' \in Q_{i,M}^T{}'$	0	0	0	$\Theta(q_{i,j}^T, \sigma)$
$q \in Q_{REST}$[4]	0	0	$*$	$*$

[1] $\Sigma_{REST} = \Sigma_C' \setminus \{\text{START}, \text{STOP}, \sigma_i^{CMD}\}$
[2] $*$ stands for unchanged, i.e. $\Theta'(q,\sigma) = \Theta(q,\sigma)$
[3] ow = otherwise
[4] $Q_{REST} = Q' \setminus \{\{q_{i,0}^T, q_{i,0}^T{}'\} \cup Q_{i,m}^T \cup Q_{i,m}^T{}'\}$

Proof: Let the language of the task T_i denoted by $L(T_i) = \{s \in \Sigma^* | \rho(q_{i,0}^T, s) \in Q_i^T\}$, while the set of strings leading to one of the final states $q_{i,M}^T \in Q_{i,M}^T$ of the task by $L_M(T_i) = \{s \in \Sigma^* | \rho(q_{i,0}^T, s) \in Q_i^T\}$. Therefore, the language generated by the supervised plant is $L(S/G) = \{t.\{L(T_i).u | t, u \in \Sigma^*, \exists \rho(q_0, t.L(T_i).u)\}$.

Let the resulting system of the extension, composed as the synchronous product of the plant and the watchdog, denoted by $G' = G \| WD$. Since the event sets of G and WD are distinct, i.e. $\Sigma^G \cap \{\text{START}, \text{STOP}, \text{ALARM}, \text{RESET}\} = \emptyset$, the language generated by the plant G is the natural projection of the language generated by the extended system to the event set of G: $P_{\Sigma^G}(L(S'/G')) = P_{\Sigma^G}(t'.L(T_i').u')$.

According to the properties of the natural projection, $P_{\Sigma^G}(t'.L(T_i').u') = P_{\Sigma^G}(t').P_{\Sigma^G}(L'(T_i)).P_{\Sigma^G}(u')$. Due to its definition, the extension effects only the language of the task, so $t' = t$ and $u' = u$, therefore $P_{\Sigma^G}(t') = t$ and $P_{\Sigma^G}(u') = u$. Thus, $P_{\Sigma^G}(L(S'/G')) = t.P_{\Sigma^G}(L(T_i')).u$.

The language of the extended task T_i' is defined as follows:

$$\begin{aligned} L(T_i') = & \ \text{START}.L(T_i) \\ & \cup \text{START}.L_M(T_i).\text{STOP} \\ & \cup \{\text{START}.L(T_i) \setminus \{\{\epsilon\} \cup L_M(T_i)\}\}.\text{ALARM}.v \\ & | v \in \Sigma^* \text{ and } \exists \rho(\xi(T_i), v) \end{aligned}$$

Since we assume that no fault occurs, it can be restricted to $L_{\mathrm{NF}}(T_i') = \textsc{start}.L(T_i) \cup \textsc{start}.L_M(T_i).\textsc{stop}$, so

$$P_{\Sigma^G}(L_{\mathrm{NF}}(T_i')) = \begin{array}{l} P_{\Sigma^G}(\textsc{start}.L(T_i)) \\ \cup P_{\Sigma^G}(\textsc{start}.L_M(T_i).\textsc{stop}) \end{array}$$

Therefore, $P_{\Sigma^G}(L_{\mathrm{NF}}(L(T_i')) = L(T_i)$, so

$$P_{\Sigma^G}(L(S'/G'))=t.P_{\Sigma^G}(L_{\mathrm{NF}}(T_i')).u = t.L(T_i).u = \\ =L(S/G)$$

■

Remark: If we assume that any fault occurs, no such proposition can be made. Since a fault triggers the ALARM event, the system will execute an alarm handling procedure, which usually differs significantly from the normal operation of the system.

V. Fault detection in distributed environment

When dealing with large-scale and more sophisticated systems, one of the frequently used solutions to avoid the state-explosion problem is to use distributed control structures (see, for example, [11]). However, a malfunction in a given subsystem can cause the failure of other subsystem(s), so fault detection in distributed environments has a paramount importance.

The common situation is that a subsystem, G_2, needs some resources provided by a remote subsystem G_1 for its operation. Assume that there is a watchdog associated to G_1, so its controller, C_1 is informed about the faults of G_1. However, since a fault in G_1 can effect the operation of G_2, even causing a dangerous situation, it is vital to provide a possibility for C_2 to check whether a failure has occurred in the remote subsystem, namely G_1.

In order to allow controllers to gain information about the failures of remote subsystems, a suitable communication should be found to ensure the information exchange between the controller and the watchdog associated to the remote subsystem. The query-response philosophy, illustrated in Fig. 4 provides a low-cost solution, and can be implemented by carrying out a few extensions on the model of the watchdog, needing no additional components.

A. Extended model of the watchdog

To implement communication functions, the event set of the watchdog should be extended by the controllable query and uncontrollable response events, QUERY, R_IDLE and R_ALARM. The query event is used by the controllers to sign their request for information, while the response events are

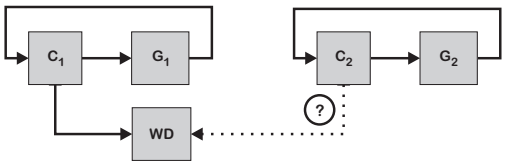

Fig. 4. Watchdog in distributed environment

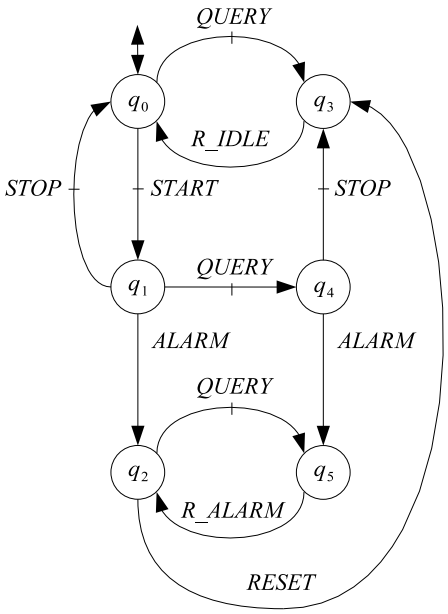

Fig. 5. Communication enabled extended model of the watchdog

generated by the watchdog reporting its actual state, indicating whether a fault has occurred in the guarded subsystem. Note that when the watchdog is in its $Running$ state, there is no reliable information available on the functionality of the guarded subsystem. It can be stopped in the next moment, indicating no failure, or it can also pass to its $Alarm$ state, indicating the presence of a fault.

The extended model of the watchdog is depicted in Fig. 5. The model is extended by three new, query states, namely $Idle\_q$ (q_3), $Running\_q$ (q_4) and $Alarm\_q$ (q_5), which are reached when the watchdog receives a query in the corresponding state. We assume that the implementation of the watchdog is such that in $Idle\_q$ and $Alarm\_q$ states, the corresponding response events are generated instantaneously, leading the watchdog back to the $Idle$ and $Alarm$ states, respectively. Note that, according to the principle described in the previous paragraph, no response event is generated immediately when a query is received in the $Running$ state. In that case, the corresponding response events follow the STOP or ALARM events. An R_IDLE response event is generated, even if the watchdog has not yet been queried, upon the reset, which feature will be used in the sequel.

B. Wait-for-OK strategy

It is a common situation that a subsystem needs some resources provided by an other one to execute a task, so the given operation can not be carried out when the other subsystem is failed. Likely, there are cases when starting a task when an other subsystem is down can result in damage or even injuries. For example, when two conveyors are situated one following the other, the first one should not be started when the second one is down in order to avoid the stuck of workpieces, and therefore damage of valuable

material.

For these situations the Wait-for-OK strategy can be used. Let us assume that G_2 needs the resources of G_1, which is under the guard of a watchdog, for executing a given task. Before starting the task, C_2 should query the watchdog associated to G_1, and continue its operation only if the watchdog is found in its $Idle$ state, i.e. the R_IDLE response event is generated. If the R_ALARM response event is generated, indicating that a failure has occurred in G_1, the only solution for C_2 is to suspend its operation and wait for the handling of the fault. Since the event R_IDLE is generated upon the reset of the watchdog, G_2 can continue its operation after the handling of the failure.

If it has not yet been done, at first the controller model of the remote subsystem, C_1 should be extended by following the method presented in Section IV-B in order to incorporate watchdog-based fault detection functions. Let $C_2 = \{Q_2, \Sigma_2, \rho_2, q_{2,0}, Q_{2,M}\}$ denote the controller of G_2 and $C_2' = \{Q_2', \Sigma_2', \rho_2', q_{2,0}', Q_{2,M}\}$ its extension. The state of the controller from where the given operation is started will be denoted by q_j. To use the wait-for-OK strategy, two new states, q_j' and q_j'' should be added to the state set of C_2, so $Q_2' = Q_2 \cup \{q_j', q_j''\}$, while the event set should be extended by the QUERY and R_IDLE events, so $\Sigma_2' = \Sigma_2 \cup \{\text{QUERY,R\_IDLE}\}$. The transition function should be extended to ρ_2' for $\forall q \in Q_2'$ and $\forall \sigma \in \Sigma_2'$ by the followings:

$$\rho_2'(q,\sigma) = \begin{cases} \rho_2(q,\sigma) & \forall q \in Q_{\setminus}\{q_j\}, \forall \sigma \in \Sigma_2 \\ q_j' & \text{iff } q = q_j, \sigma = \text{QUERY} \\ q_j'' & \text{iff } q = q_j' \sigma = \text{R\_IDLE} \\ \rho_2(q_j,\sigma) & \forall \sigma \in \Sigma_2, q = q_j'' \\ \text{undefined otherwise} \end{cases}$$

For the definition of the initial state of C_2', the following rule should be applied:

$$q_{2,0}' = \begin{cases} q_j' & \text{if } q_{2,0} = q_j \\ q_{2,0} & \text{otherwise} \end{cases}$$

The extension of the controller is illustrated by Fig. 6, while the modification of the control map is given by Table II.

Proposition 2: Using the Wait-for-OK strategy, if no failure is detected by the watchdog, i.e. no ALARM event is generated, the plants G_1 and G_2 act the same under the supervision of the original and extended controllers by the mean that the languages they generate are not effected by the extensions, so that $P_{\Sigma_1^G}(L(S_1'/G_1')) = L(S_1/G_1)$ and $P_{\Sigma_2^G}(L(S_2'/G_2')) = L(S_2/G_2)$.

Proof: According to Proposition 1, the language generated by G_1 is not effected by the extension, so $P_{\Sigma_1^G}(L(S_1'/G_1')) = L(S_1/G_1)$.

The equivalence of the languages generated by G_2 is straightforward. ■

TABLE II

MODIFICATION OF THE CONTROL MAP IN CASE OF USING THE WAIT–FOR–OK STRATEGY

	$QUERY$	$\sigma \in \Sigma_{2,C}' \setminus \{QUERY\}$
q_j	0	$\Theta_2(q_j,\sigma)$
q_j'	1	$\begin{cases} 0 & \text{if } \exists\rho(q_j,\sigma) \\ \Theta_2(q_j,\sigma) & \text{otherwise} \end{cases}$
q_j''	0	$\begin{cases} 0 & \text{if } \exists\rho(q_j,\sigma) \\ \Theta_2(q_j,\sigma) & \text{otherwise} \end{cases}$
$q \in Q_{REST}$[1]	0	$\Theta_2(q,\sigma)$

[1] $Q_{REST} = Q_2{}' \setminus \{q_j, q_j', q_j''\}$

C. Multimodal strategy

More sophisticated subsystems, initially using some resources provided by remote components, are often capable of switching to a degraded mode, in which they can continue their operation without those resources. For example, a robot arm, placing workpieces on a conveyor, can depose the pieces to a temporary buffer in case of the failure of the conveyor.

When dealing with subsystems having more operational modes, the approach proposed by Kamach will be used [12], [13]. Identical process and specification models are constructed for each operational mode, based on which supervisors are synthesized. Controller models include a so-called inactive state (q_{IA}), to where the controller model passes enters upon commuting to another operational mode. The newly activated controller model is activated by passing from its inactive state to its starting state (q_{S}).

Like in the case of the Wait-for-OK strategy, the controller of G_2 shall query the watchdog associated to G_1 before starting the operation of G_2 needing the resources provided by G_1. If the watchdog is found to be in its $Alarm$ state, the subsystem should be switched to its degraded mode, in which it can continue its operation without the resources provided by G_1. In the degraded mode, C_2 should query the watchdog in every duty cycle and switch back to nominal mode immediately, if the failure has been handled, i.e. the watchdog associated to G_1 is found to be in its $Idle$ state.

Let us assume again that G_1 is equipped with a watchdog, and C_1 has been extended to incorporate watchdog-based fault detection capabilities. The nominal mode, needing resources provided by G_1, will be denoted by M_1,

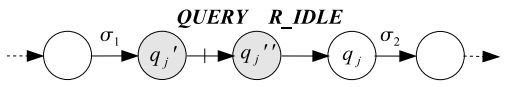

Fig. 6. Extension of the controller model using the Wait-for-OK strategy

TABLE III

EXTENSION OF THE CONTROL MAP OF C_2^1 USING THE MULTIMODAL STRATEGY

	QUERY	$\sigma \in {\Sigma_{2C}^1}' \setminus \{\text{QUERY}\}$
q_j	0	$\Theta_2^1(q_j, \sigma)$
q_j'	1	$\begin{cases} 0 & \text{if } \exists \rho_2^1(q_j, \sigma) \\ \Theta(q_j, \sigma) & \text{otherwise} \end{cases}$
q_j''	0	$\begin{cases} 0 & \text{if } \exists \rho(q_j, \sigma) \\ \Theta(q_j, \sigma) & \text{otherwise} \end{cases}$
$q \in Q_{\text{REST}}{}^1$	0	$\Theta_2^1(q, \sigma)$

[1] $Q_{\text{REST}} = {Q_2^1}' \setminus \{q_j, q_j', q_j''\}$

TABLE IV

EXTENSION OF THE CONTROL MAP OF C_2^2 USING THE MULTIMODAL STRATEGY

	QUERY	$\sigma \in {\Sigma_{2C}^2}' \setminus \{\text{QUERY}\}$
q_n	0	$\Theta_2^2(q_n, \sigma)$
q_n'	1	$\begin{cases} 0 & \text{if } \exists \rho_2^2(q_n, \sigma) \\ \Theta_2^2(q_n, \sigma) & \text{otherwise} \end{cases}$
q_n''	0	$\begin{cases} 0 & \text{if } \exists \rho_2^2(q_n, \sigma) \\ \Theta_2^2(q_n, \sigma) & \text{otherwise} \end{cases}$
$q \in Q_{\text{REST}}{}^1$	0	$\Theta_2^2(q, \sigma)$

[1] $Q_{\text{REST}} = {Q_2^2}' \setminus \{q_n, q_n', q_n''\}$

while the degraded mode will be referred to as M_2. Controller models designed for the nominal and degraded modes are given by $C_2^1 = \{Q_2^1, \Sigma_2^1, \rho_2^1, q_{2,0}^1, Q_{2,M}^1\}$ and $C_2^2 = \{Q_2^2, \Sigma_2^2, \rho_2^2, q_{2,0}^2, Q_{2,M}^2\}$, respectively.

The controller model C_2^1 of the nominal mode should be extended to ${C_2^1}' = \{{Q_2^1}', {\Sigma_2^1}', {\rho_2^1}', {q_{2,0}^1}', Q_{2,M}^1\}$ by the followings. Three new states, q_j', q_j'' and q_{IA}^1 should be added to the state set, so that ${Q_2^1}' = Q_2^1 \cup \{q_j', q_j'', q_{IA}^1\}$. It is assumed that an existing state has been already chosen as the starting state, so $q_S^1 \in Q_2^1$. The query and response events should be also introduced, so the new event set will be ${\Sigma_2^1}' = \Sigma_2^1 \cup \{\text{QUERY}, \text{R\_IDLE}, \text{R\_ALARM}\}$. The transition function should be extended to ${\rho_2^1}'$ for $\forall q \in {Q_2^1}'$ and $\forall \sigma \in {\Sigma_2^1}'$ as

$$
{\rho_2^1}'(q, \sigma) = \begin{cases} \rho_2^1(q, \sigma) & \forall q \in Q_2^1 \backslash \{q_j\}, \forall \sigma \in \Sigma_2^1 \\ q_j' & \text{for } q = q_j, \sigma = \text{QUERY} \\ q_j'' & \text{for } q = q_j', \sigma = \text{R}_{\text{IDLE}} \\ \rho_2^1(q_j, \sigma) & \text{for } q = q_j'', \sigma \in \Sigma_2^1 \\ q_{IA}^1 & \text{for } q = q_j', \sigma = \text{ALARM} \\ q_S^1 & \text{for } q = q_{IA}^1, \sigma = \text{R}_{\text{IDLE}} \\ \text{undefined otherwise} \end{cases}
$$

For the definition of initial state of ${C_2^1}'$, the following rule should be applied:

$${q_{2,0}^1}' = \begin{cases} q_j' & \text{if } q_{2,0}^1 = q_j \\ q_{2,0}^1 & \text{otherwise} \end{cases}$$

The control map Θ_2^1 should be extended to ${\Theta_2^1}'$ according to Table III .

Similarly, the controller model C_2^2 of the degraded mode should be extended to ${C_2^2}' = \{{Q_2^2}', {\Sigma_2^2}', {\rho_2^2}', {q_{2,0}^2}', Q_{2,M}^2\}$. Three new states, q_n', q_n'' and q_{IA}^2 should be added to its state set, so that ${Q_2^2}' = Q_2^2 \cup \{q_n', q_n'', q_{IA}^2\}$. It is assumed that an existing state ha already chosen as the starting state, so $q_S^2 \in Q_2^2$. The query and response events should also be introduced, so the resulting event set of ${C_2^2}'$ will be ${\Sigma_2^2}' = \Sigma_2^2 \cup \{\text{QUERY}, \text{Q\_IDLE}, \text{R\_ALARM}\}$. The transition

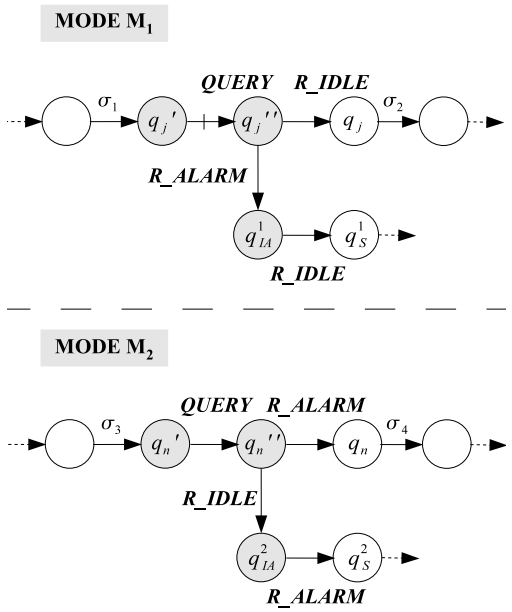

Fig. 7. Extension of the controller models using the multimodal strategy

function should be extended to ${\rho_2^2}'$ for $\forall q \in {Q_2^2}'$ and $\forall \sigma \in {\Sigma_2^2}'$ according to the followings:

$$
{\rho_2^2}'(q, \sigma) = \begin{cases} \rho_2^2(q, \sigma) & \forall q \in Q_2^2 \backslash \{q_n\}, \forall \sigma \in \Sigma_2^2 \\ q_n' & \text{for } q = q_n, \sigma = \text{QUERY} \\ q_n'' & \text{for } q = q_n', \sigma = \text{R}_{\text{ALARM}} \\ q_n & \text{for } q = q_n'', \sigma \in \Sigma_2^2 \\ q_{IA}^2 & \text{for } q = q_n', \sigma = \text{R}_{\text{IDLE}} \\ q_S^2 & \text{for } q = q_{IA}^2, \sigma = \text{R}_{\text{ALARM}} \\ \text{undefined otherwise} \end{cases}
$$

For the definition of the initial state of ${C_2^2}'$ the following rule should be applied:

$${q_{2,0}^2}' = \begin{cases} q_n' & \text{if } q_{2,0}^2 = q_n \\ q_{2,0}^2 & \text{otherwise} \end{cases}$$

The control map Θ_2^2 should be extended to $\Theta_2^{2'}$ according to Table IV. The extension of the controller models is illustrated by Figure 7.

Proposition 3: Using the multimodal strategy, if no failure is detected by the watchdog, i.e. no ALARM event is generated, the processes G_1 and G_2 act the same under the supervision of the original and the extended controllers, by the mean that the languages they generate are not effected by the extensions, so $P_{\Sigma_1^G}(L(S_1'/G_1')) = L(S_1/G_1)$ and $P_{\Sigma_2^G}(L(S_2'/G_2')) = L(S_2/G_2)$ for both operational modes.

The proof of the first part is evident, while the proof of the second part is analogous to the method used in case of Proposition 1.

VI. Conclusion

The approach presented in this paper places well-known watchdog structures in the SCT framework. The presented methods allow the extension of previously designed controllers to integrate watchdog-based fault detection techniques both in centralized and distributed supervisory control environments.

The definition of discrete-event models of the watchdog and the formal description of the extension methods, along with their systematic properties, allow the implementation of algorithms. This allow formal integration of fault detection capabilities that helps system designers to realize low-cost fault detection methods in a systematic way.

The integration of the presented methods into rapid control prototyping environments, allowing simple modeling, simulation and implementation on various hardware platforms may be the next step towards the use of the concept in industrial applications. Problems of observability and diagnosability may be also studied using formalism provided by the SCT framework.

Acknowledgments

This research was partially funded by the Hungarian National Office for Research and Technology under grant OMFB-01418/2004 (Advanced Vehicle and Vehicle Control Knowledge Center).

References

[1] G. Isermann, "Model-based fault-detection and diagnosis – status and applications," *Annual Reviews in Control*, vol. 29, pp. 71–85, 2005.

[2] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Boston: Kluwer Academic Publishers, 1999.

[3] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Failure diagnosis using discrete-event models," *IEEE Trans. Control Systems Technology*, vol. 48, pp. 105–120, 1996.

[4] Y. Ting, F. Shan, W. Lu, and C. Chen, "Implementation and evaluation of failsafe computer-controlled systems," *Computers & Industrial Engineering*, vol. 42, pp. 401–415, 2002.

[5] S. Zad, R. Kwong, and W. Wonham, "Fault diagnosis in discrete-event systems: Framework and model reduction," *IEEE Trans. Automatic Control*, vol. 48, pp. 1199–1211, 2003.

[6] O. Contant, S. Lafortune, and D. Teneketzis, "Diagnosis of intermittent faults," *Discrete Event Dynamic Systems*, vol. 14, pp. 171–202, 2004.

[7] W. Wonham, *Notes on Control of Discrete Event Systems*. University of Toronto, 2002.

[8] R. Kumar, V. Garg, and S. Marcus, "On controllability and normality of discrete event systems," *Systems & Control Letters*, vol. 17, pp. 157–168, 1991.

[9] R. Brandt, V. Garg, R. Kumar, F. Lin, S. Marcus, and W. Wonham, "Formulas for calculating supremal controllable and normal sublanguages," *System & Control Letters*, vol. 15, pp. 157–168, 1990.

[10] L. Holloway and B. Krogh, "Fault detection and diagnosis in manufacturing systems: A behavioral model approach," *Proc. Second International Conference on Computer Integrated Manufacturing*, vol. 1, pp. 252–259, 1990.

[11] M. Nourelfath and E. Niel, "Modular supervisory control of an experimental automated manufacturing system," *Control Engineering Practice*, vol. 12, pp. 205–216, 2004.

[12] O. Kamach, S. Chafik, L. Piétrac, and E. Niel, "Representation of a reactive system with different models," *Proc. IEEE International Conference on Systems*, vol. 4, pp. 263–267, 2002.

[13] O. Kamach, L. Piétrac, and E. Niel, "Multi-model approach to discrete event systems: Application to operating mode management," *Mathematics and Computers in Simulation*, vol. 70, pp. 396–407, 2006.